

마이크로프로세서응용

기말 프로젝트 보고서

조번호: 05

소속: 전산전자공학부 학번: 21900543 이름: 이원빈

소속: 전산전자공학부 학번: 21900549 이름: 이윤서

과목명: 마이크로프로세서응용

담당교수명: 이 강

제출일: 2024.06.25

목차

1 프로젝트 개요	3
1.1 문제 정의	3
1.2 사용자 인터페이스 정의	3
1.3 사용된 입출력 장치 목록	3
2. 설계 및 구현	4
2.1 개발환경 설명	4
(1) 하드웨어 환경 (마이크로프로세서 및 보드)	4
(2) 탑재된 시스템 소프트웨어 (RTOS 및 주요 라이브러리)	4
(3) 개발에 사용된 언어 및 IDE	4
(4) 사용된 주변장치 상세 내역	4
(5) CMakeList.txt	5
2.2 프로그램의 구조	6
(1) 흐름도	6
(2) 모듈 설정 (Kconfig)	7
2.2 주변장치	7
(1) 사용된 메모리맵 (주변장치별 접근 주소)	7
(2) 디바이스 트리 (devicetree overlay)	8
(3) 주변장치별 프로세서와 통신 표준 설명	10
2.3 프로세스 및 인터럽트 관리	10
(1) 인터럽트 설정 (우선순위, ISR 등)	10
(2) Multi-Thread 사용시 Thread, 별 역할과 통신	11
3. 구현 결과	11
3.1 사용방법 설명	11
3.2 동작시연 (이미지와 더불어 설명)	11
3.3 자체 평가	13
(1) 난이도 평가 (평가 기준에 근거하여 제시)	13
(2) 계획대비 완성도	13
(3) 조원의 개인 기여 내용	13
4. 부록	13
4.1 작성된 파일 목록 및 설명 (프로젝트 README 파일 내용)	13
4.2 개발과정에서 만났던 에러가 있는 경우, 에러메시지와 그 대처 방안 설명	13
4.3 개인 소감	13
4.4 개발과정에서 참조 했던 리소스	14

1. 프로젝트 개요

1.1 문제 정의

현대 사회에서 디지털 게임은 많은 사람들의 여가 활동에 있어서 중요한 부분을 차지하고 있다. 매우 다양한 종류의 게임이 개발되어 있으며, 특히 스마트폰의 성능향상에 따라서 스마트폰을 통해 언제 어디서나 게임을 즐길 수 있다. 그러나 대부분의 스마트폰 디지털 게임은 매우 복잡한 게임방식과 자극적이고 화려한 그래픽 인터페이스를 제공하는 경우가 많아 큰 시각적 자극 없이 간단하게 즐길 수 있는 게임을 찾기 어려운 경우가 많다. 더욱이 청소년들이 스마트폰을 사용한 게임에 익숙해지면서 과도한 전자기기 사용시간의 증가로 이어지는 경우도 흔하게 발생하게 된다. 이러한 문제를 해결하기 위해 단순한 인터페이스와 조작법으로 누구나 편하게 즐길 수 있는 자동차 경주 게임기를 개발하고자 했다. 뿐만 아니라 다른 전자기기들과 비교할 때 해당 게임기의 구매 비용을 매우 낮추고, 성능이 낮은 하드웨어에서도 구동될 수 있는 게임을 개발하고자 하였으며, 단순한 게임을 넘어서 다양한 센서들을 사용해 컴퓨터와 상호작용하는 경험을 제공하고, 간단한 환경정보 등을 제공받을 수 있는 프로젝트를 계획하였다.

1.2 사용자 인터페이스 정의

사용자 인터페이스를 최대한 간단하고 직관적으로 설계해 다양한 입력 및 출력 장치를 통해 사용자가 시스템과 상호작용할 수 있도록 했다. 입력 장치로는 4개의 푸시 버튼이 있으며, 각각 게임 시작, CO2 측정 모드 진입, 소리 감지 모드 진입, 보드 상태 확인 기능을 수행한다. 또한, 소리 감지 센서는 소리 감지 모드에서 주위 소음을 감지하고, 조이스틱은 게임 진행 중 자동차의 움직임을 제어하며, 로터리 인코더는 화면 밝기를 조절한다. 출력 장치로는 LED 매트릭스가 게임 화면과 환경 정보를 시각적으로 표시하고, 배터리 디스플레이는 게임 진행 정도를 나타내며, 부저는 게임 시작 시 효과음을 출력한다. 메인 화면에서는 "WAIT" 메시지가 출력되며, 사용자가 입력을 대기한다. 게임 시작 화면에서는 카운트다운과 함께 시작 신호음이 울리고, 게임 화면에서는 조이스틱 입력에 따라 자동차가 움직이며 배터리 디스플레이에 진행 정도가 표시된다. CO2 측정 모드에서는 CO2 수치가 LED 매트릭스에 시각적으로 출력되고 3초 후 메인 화면으로 복귀하며, 소리 감지 모드에서는 소리 크기가 LED 매트릭스에 표시되고 소리가 감지되지 않으면 X 표시가 나타난다. 어떤 상태에서든지 로터리 인코드를 사용해 화면의 밝기를 조정할 수 있다. 이 모든 요소는 사용자가 시스템을 쉽게 이해하고 조작할 수 있도록 하여 높은 사용성을 제공한다.

1.3 사용된 입출력 장치 목록

1. **Push Button (총 4개)** : 메인 화면에 있을 때, 디바이스에서 제공하는 여러 모드를 실행하거나 종료하는데 사용
2. **Sound Sensor** : 사용자가 Sound Sensing 모드를 실행했을 때, 주위의 소음을 감지하는데 사용
3. **Joy Stick** : 게임을 진행하는 동안 자동차의 움직임을 입력받는데 사용
4. **Rotary Encoder** : 화면의 밝기를 조정하는데 사용
5. **CO2 sensor** : CO2 농도를 측정하는데 사용
6. **LED Matrix** : 게임화면이나 측정된 환경 값들을 사용자에게 출력하는데 사용
7. **Battery Display** : 게임 진행 정도를 출력하는데 사용
8. **Buzzer** : 게임 시작시 효과음을 출력하는데 사용

2. 설계 및 구현

2.1 개발환경 설명

(1) 하드웨어 환경 (마이크로프로세서 및 보드)

Cortex-m4 CPU가 탑재된 nrf52840보드를 사용하였고, 해당 마이크로컨트롤러에 Open-Smart Rich Shield Two보드를 부착하여 사용하였다.

(2) 탑재된 시스템 소프트웨어 (RTOS 및 주요 라이브러리)

Zephyr Real Time OS를 탑재하여 사용하였으며, OS에서 제공하는 다음과 같은 라이브러리들을 사용하였다.

- bluetooth.h : BLE통신을 위한 라이브러리
- drivers/adc.h : 아날로그 신호를 사용하는 sound sensing과 joystick 사용을 위한 라이브러리
- drivers/uart.h : uart통신을 사용하는 co2 sensing을 위한 라이브러리

(3) 개발에 사용된 언어 및 IDE

모든 소스코드 파일들은 C 언어로 작성되었으며, nRF Connect SDK와 VSCode 코드에디터에서 제공하는 nRF Connect Extension을 사용해 보드를 연결하여 프로젝트를 진행하였다.

(4) 사용된 주변장치 상세 내역

1. CO2 센서

CO2 센서는 MH-Z14A센서를 사용하였다. 해당 장치는 UART통신과 serial port를 사용해 측정된 공기 중 CO2 농도를 감지한 데이터를 전송한다. 보드와의 연결을 위해서 모듈의 Vin, GND, RXD, TXD를 각각 보드의 5V, GND, RXD, TXD에 연결해주어야 한다. 해당 프로젝트에서는 Rich Shield 보드에 연결해 사용했다.

2. 수동부저

수동부저는 능동부저와 달리 주파수에 맞는 신호가 입력되면 그에 맞는 소리를 출력한다. 따라서 nrf52840보드의 PWM 출력 핀에 연결해 PWM신호를 전달해 제어했다. 수동부저의 +핀은 보드의 GND핀과 연결했으며, -핀은 보드의 P0.13핀과 연결했다. P0.13핀은 zephyr의 Device Tree에서 LED1 을 제어하도록 설정되어있는 PWM핀이다. 해당 프로젝트에서는 점프선을 사용해 부저를 P0.13핀과 연결해 PWM신호로 LED1과 부저가 함께 제어되도록 설계했다.

3. LED Matrix

Rich Shield에 부착되어 있는 LED Matrix를 사용했다. Rich Shield에는 HT16K33 LED Driver IC가 존재해서 16x8 LED Matrix의 각각의 LED를 점등하고 밝기를 제어하는 등의 기능을 쉽게 수행할 수 있다. 또한 해당 드라이버는 nrf52840보드의 Inter Integrated Circuit을 통해서 I2C프로토콜을 사용해 통제된다.

4. Joystick

Rich Shield에 부착되어 있는 Joystick을 사용했다. Joystick은 x축과 y축 아날로그 값을 출력한다. 이 값을 읽고 처리하기 위해서 nrf52840의 ADC 입력 핀에 연결한다. nrf52840에서는 ADC 채널을 설정하고 입력 핀을 설정해 아날로그값을 읽어서 처리한다.

5. Sound Sensor

Rich Shield에 부착되어 있는 Sound Sensor를 사용했다. Joystick과 마찬가지로 ADC채널을 사용해서 아날로그 값을 디지털 값으로 변환해 사용했다.

6. Rotary Encoder

Rich Shield에 부착되어 있는 Rotary Encoder를 사용했다. 두개의 회전정보를 출력하는 채널과, 버튼을 위한 채널이 존재한다. 이 채널들은 각각 GPIO핀에 연결해 사용했다.

7. Battery Display

Rich Shield에 부착되어 있는 Battery Display를 사용했다. 총 7개의 레벨로 나누어 디스플레이 표현을 하였으며, GPIO 1번 포트의 12번, 13핀과 연결하여 사용하였다.

(5) CMakeList.txt

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

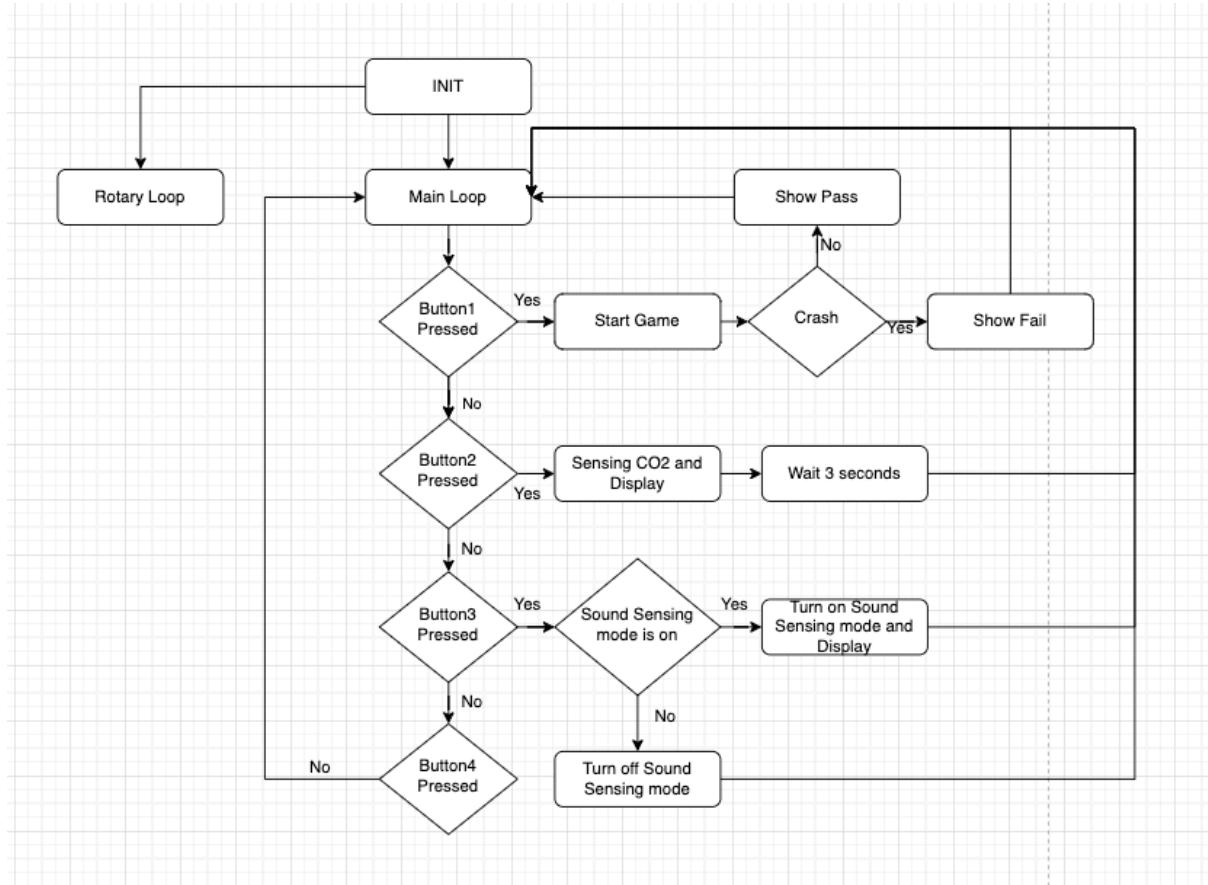
project(final_project)

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

해당 프로젝트를 빌드하기 위해서는 CMake 3.20.0버전 이상이 필요하며, Zephyr RTOS와 함께 빌드된다. 또한 src 디렉토리 내부의 모든 C파일들과 src디렉토리 내부의 map.s라는 어셈블리 파일이 타겟 파일에 포함된다.

2.2 프로그램의 구조

(1) 흐름도



(2) 모듈 설정 (Kconfig)

```
CONFIG_SENSOR=y
CONFIG_PRINTK=y

CONFIG_ADC=y
CONFIG_GPIO=y
CONFIG_UART_INTERRUPT_DRIVEN=y
CONFIG_UART_ASYNC_API=y
CONFIG_LOG=y
CONFIG_I2C=y
CONFIG_LED=y
CONFIG_PWM=y
CONFIG_KSCAN=y
CONFIG_KSCAN_INIT_PRIORITY=95
CONFIG_HT16K33_KEYSCAN=y

CONFIG_USE SEGGER RTT=y
CONFIG_RTT_CONSOLE=y

CONFIG_BT=y
CONFIG_BT_DEBUG_LOG=y
CONFIG_BT_L2CAP_TX_BUF_COUNT=5
CONFIG_BT_PERIPHERAL=y

CONFIG_BT_DEVICE_NAME="My_Device"
CONFIG_BT_DEVICE_APPEARANCE=962

CONFIG_HEAP_MEM_POOL_SIZE=2048

# This example requires more workqueue stack
CONFIG_SYSTEM_WORKQUEUE_STACK_SIZE=2048
```

2.2 주변장치

(1) 사용된 메모리맵 (주변장치별 접근 주소)

- **Buzzer** : 4001c000(pwm0)
- **CO2 Sensor** : 40028000(uart)
- **LED-Matrix** : 40003000(i2c)
- **Battery Display** : 4002f000(`3)
- **Joystick** : 40007000adc channel1,2
- **Sound Sensor** : 40007000adc channel3
- **Rotary Encoder** : 40012000(qdec)
-

(2) 디바이스 트리 (devicetree overlay)

```
zephyr,user {
    io-channels = <&adc 1>, <&adc 2>, <&adc 3>;
};

pwmbuzzer {
    compatible = "pwm-leds";
    status = "okay";

    buzzer: buzzer_pwm {
        pwms = <&pwm0 0 PWM_MSEC(880) PWM_POLARITY_NORMAL>;
        label = "PWM_1";
    };
};
```

```

gpiocustom{
    status = "okay";
    compatible = "gpio-keys";
    gpiosw: gpiosw {
        gpios = <&gpio1 5 (GPIO_PULL_UP)>;
        label = "gpiosw P1.05";
    };

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

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

aliases {
    myserial = &uart1;
    gpio-clk = &gpioclk;
    gpio-dio = &gpiodio;
    qdec0 = &qdec0;
    gpio-sw = &gpiosw;
};

```

```

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

    channel@1 {
        reg = <1>;
        zephyr,gain = "ADC_GAIN_1";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN1>; /* P0.03 */
        zephyr,resolution = <10>;
        // zephyr,differential;
    };

    channel@2 {
        reg = <2>;
        zephyr,gain = "ADC_GAIN_1";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN2>; /* P0.04 */
        zephyr,resolution = <10>;
        // zephyr,differential;
    };

    channel@3 {
        reg = <3>;
        zephyr,gain = "ADC_GAIN_1";
        zephyr,reference = "ADC_REF_INTERNAL";
        zephyr,acquisition-time = <ADC_ACQ_TIME_DEFAULT>;
        zephyr,input-positive = <NRF_SAADC_AIN4>; /* A2 */
        zephyr,resolution = <10>;
    };
};

```

```

&pinctrl {
    qdec_pinctrl: qdec_pinctrl {
        group1 {
            psels = <NRF_PSEL(QDEC_A, 1, 4)>, /* Arduino D3 */
                  <NRF_PSEL(QDEC_B, 1, 3)>; /* Arduino D2 */
        };
    };

    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, 11)>;
            low-power-enable;
        };
    };
};

```

```

&qdec0 {
    status = "okay";
    pinctrl-0 = <&qdec_pinctrl>;
    pinctrl-names = "default";
    steps = < 20 >;
    led-pre = < 500 >;
};

// For battery display

&spi3 {
    status = "disabled";
};

```

```
// For LED Matrix

&i2c0 {
    clock-frequency = <I2C_BITRATE_STANDARD>

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

        keyscan {
            compatible = "holtek,ht16k33-keyscan";
        };
    };
};

// For CO2 sensor
arduino_serial: &uart1 {
    status = "okay";
    compatible = "nordic,nrf-uart";
    current-speed = <9600>;
    pinctrl-0 = <&uart1_default>;
    pinctrl-1 = <&uart1_sleep>;
    pinctrl-names = "default", "sleep";
};
```

(3) 주변장치별 프로세서와 통신 표준 설명

1. LED Matrix - *I2C* 통신

I2C(Inter-Integrated Circuit)는 다중 마스터-다중 슬레이브, 양방향 직렬 통신 프로토콜로, 두 개의 신호 라인만으로 데이터를 전송할 수 있는 장치 간 통신 방법이다. I2C 버스는 SDA(데이터 선)와 SCL(클록 선) 두 개의 선으로 구성되며, 각 슬레이브 장치는 고유의 7비트 또는 10비트 주소를 가진다. 마스터 장치가 클록을 제공하고 슬레이브 장치들이 이에 맞추어 데이터를 전송하며, 마스터는 슬레이브의 주소를 지정한 후 읽기 또는 쓰기 명령을 통해 데이터를 전송하거나 수신한다. I2C는 표준 모드(100 kbps), 패스트 모드(400 kbps), 패스트 모드 플러스(1 Mbps), 하이스피드 모드(3.4 Mbps)로 동작할 수 있다.

2. CO2 Sensor - *UART* 통신

UART(Universal Asynchronous Receiver/Transmitter)는 직렬 통신을 위한 하드웨어 모듈로, 시작과 정지 비트를 사용하여 비동기 방식으로 데이터를 전송한다. UART는 TX(전송) 라인과 RX(수신) 라인으로 구성되며, 클록 신호를 공유하지 않고 데이터 프레임의 시작과 끝을 나타내는 비트를 사용하여 동기화를 맞춘다. 일반적으로 1 비트의 시작 비트, 59 비트의 데이터 비트, 선택적 패리티 비트, 12 비트의 정지 비트로 구성된 데이터 프레임을 사용하며, 보율(Baud rate)은 9600, 115200 등 다양한 속도로 설정 가능하다. 또한, 패리티 비트를 사용하여 전송 오류를 감지할 수 있다.

3. 핸드폰과의 연결 - *BLE* 통신

BLE(Bluetooth Low Energy)는 저전력 무선 통신 기술로, 짧은 거리에서 데이터 전송을 위해 설계되었다. BLE는 2.4 GHz ISM 대역에서 동작하며, 낮은 전력 소비를 목표로 설계되어 배터리 수명이 길다. BLE는 광고(Advertising) 모드와 연결(Connected) 모드로 동작하며, 광고 모드에서 장치가 주변에 자신의 존재를 알리고, 연결 모드에서 데이터 전송을 수행한다. BLE는 최대 1 Mbps의 속도로 데이터를 전송할 수 있으며, 다양한 응용을 지원하기 위해 여러 표준 프로파일(예: 심박수, 온도 센서)을 제공한다. 또한, 페어링과 암호화를 통해 보안 통신을 지원한다.

2.3 프로세스 및 인터럽트 관리

(1) 인터럽트 설정 (우선순위, ISR 등)

본 프로젝트에서는 총 6개의 인터럽트가 사용되었다.

1. Button Interrupt(4개)

nrf52840에 있는 총 4개의 푸시버튼을 모두 사용했으며, 각각의 버튼은 눌렸을 때 인터럽트를 발생시킨다. 버튼4는 busy flag와 상관없이 언제든 작동하는 인터럽트이며, led4를 토글한다. 반면 버튼 1,2,3은 서로 동시에 진행될 수 없으며 busy flag가 0일 때만 실행된다.

2. CO2 Interrupt

버튼 2가 눌렸을 때 보드는 CO2센서로 CO2값을 요청하게 되고, CO2센서가 보낸 값을 수신했을 때 interrupt가 발생한다. 해당 interrupt가 발생했을 때 화면에 CO2값을 LED Matrix를 통해 직관적으로 출력하게 된다.

3. Rotary Encoder Button Interrupt

Rotary encoder의 중앙 버튼을 누르게 되면 LED Matrix의 밝기를 프로그램 초기값으로 설정하는 interrupt가 발생한다. LED Matrix 밝기의 초기값은 30으로, INIT_BRIGHTNESS으로 led.h에 선언되어있다.

각 인터럽트들은 동일한 우선순위를 가지고 있으며, 하나의 인터럽트 핸들러가 실행중이라는 사실을 flag를 통해서 표시해 하나의 인터럽트를 처리할 때 다른 인터럽트가 발생하지 못하도록 했다.

(2) Multi-Thread 사용시 Thread별 역할과 통신

본 프로젝트에서는 총 3개의 Thread가 사용되었다.

1. Main Thread

프로그램의 시작과 함께 실행되는 Thread로 configuration작업을 수행한 이후 main loop를 실행해 메인 화면에서 사용자의 입력을 받고, 사용자 입력에 따른 기능을 수행하는 작업을 담당한다.

2. Joystick Thread

Joystick Thread는 Main Thread에서 사용자 입력을 받은 결과로 게임을 실행하게 되었을 때 실행된다. 게임이 진행되는 동안 Main Thread는 시간에 따른 Map을 구성해 자동차의 위치와 함께 출력하고, 충돌 여부를 판단한다. 이와 동시에 Joystick Thread에서는 조이스틱 y축 값을 실시간으로 읽어오며, 조이스틱 정중앙값 기준으로 변화의 여부가 있는지 판단하고, upward인지 downward인지에 대한 값을 Main Thread에 반환한다. Main thread는 해당 값을 읽어와 자동차의 위치를 바꾼다.

3. Rotary Thread

Rotary thread는 프로그램의 시작과 함께 실행되며 하드웨어의 전원이 꺼지기 전까지 Rotary Encoder로 들어오는 입력값을 읽어서 그 값에 따라 화면의 밝기를 조절하는 역할을 한다.

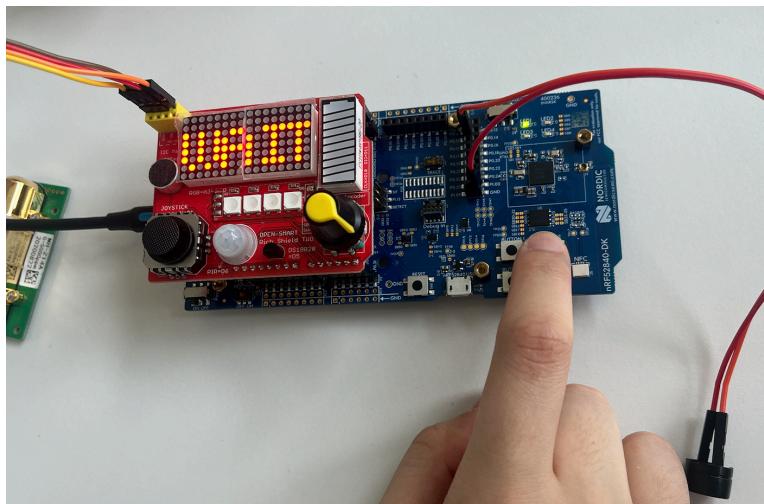
3. 구현 결과

3.1 사용방법 설명

프로그램을 빌드한 후 보드에 Flash하면, LED Matrix에 메인 화면이 나타난다. 로터리 인코더를 반시계방향으로 회전하면 화면 밝기를 어둡게 할 수 있고, 시계방향으로 회전하면 밝게 할 수 있다. 또한 로터리 인코더 중앙의 버튼을 클릭하면 밝기가 기본값으로 설정된다.

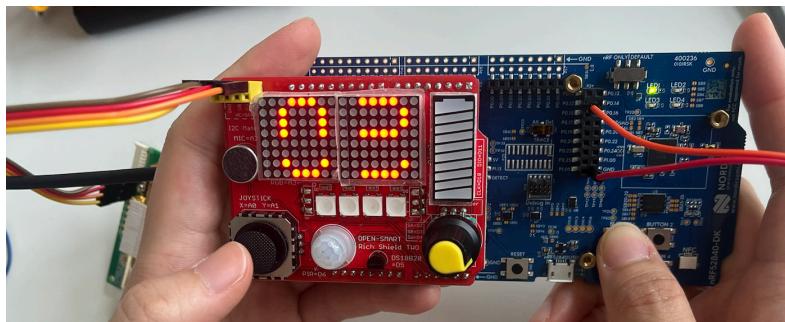
3.2 동작시연

1. 메인 화면



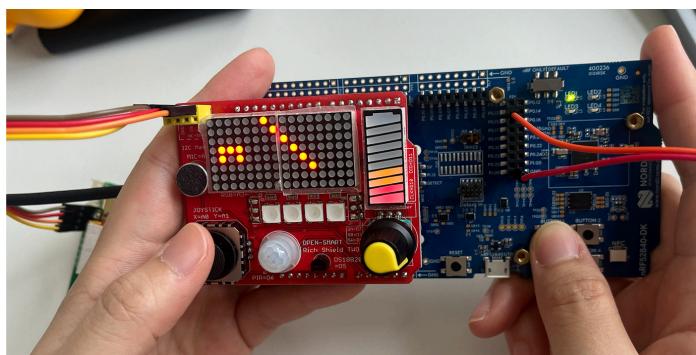
처음 게임기의 전원을 켜면 위의 사진과 같은 메인 화면이 등장한다. LED Matrix에는 WAIT이라는 문구가 출력되며 사용자의 입력이 들어오기를 기다린다. 이 상태에서 사용자가 할 수 있는 행동은 4가지가 있다. 첫번째는 버튼 1을 눌러 게임을 시작하는 것, 두번째는 버튼 2를 눌러 현재 대기중 CO₂값을 측정하는 것, 세번째는 버튼 3을 눌러 Sound Sensing Mode를 시작하는것, 마지막은 버튼 4를 눌러 현재 보드의 상태를 확인하는 것이다.

2. 게임 시작 화면



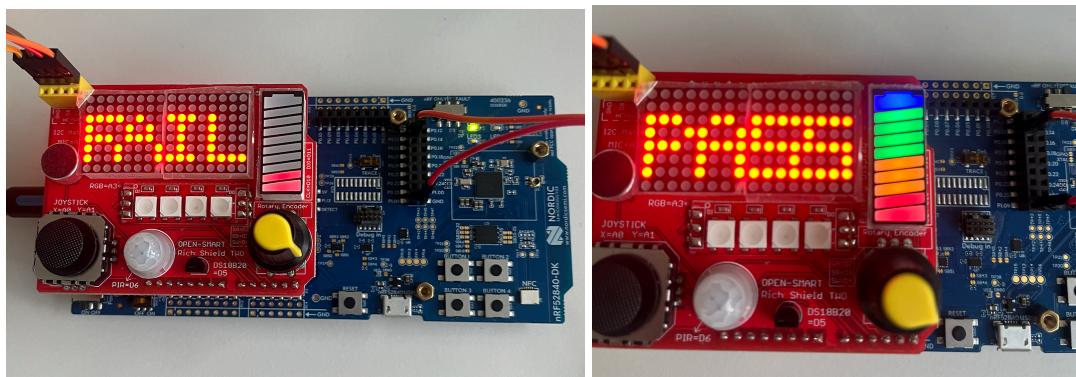
만약 사용자가 버튼 1을 눌러 게임을 시작했다면 위의 사진과 같은 게임 시작 화면이 등장한다. 화면에는 숫자 3부터 1까지 1초씩 줄어들며 출력되고, 부저에서는 숫자가 줄어드는 타이밍에 맞게 레이싱 시작 준비 신호음을 출력한다.

3. 게임 화면

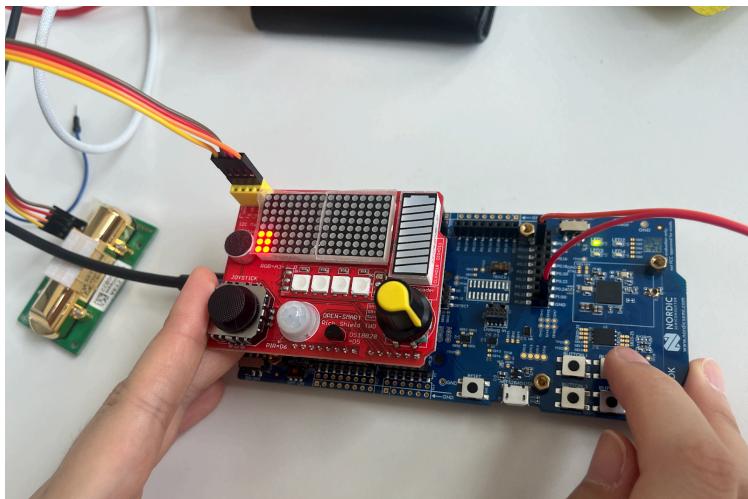


게임 시작화면이 끝나고 나면 게임화면이 출력되며 사용자의 Joystick을 통한 입력을 받아들여 화면에서 자동차의 움직임을 제어하게 된다. 게임이 진행됨에 따라 화면 우측의 배터리 디스플레이에서는 진행정도가 배터리가 충전되는 모습으로 출력되게 된다.

만약 자동차가 도착지에 도착하기 전에 장애물과 충돌하면 FAIL이라는 문구가 5초간 점등된다. 또한 자동차가 도착지에 도착하게 되면 PASS라는 문구가 5초간 점등된다. FAIL과 PASS 모두 5초가 지난 이후에는 WAIT라는 문구가 보이며 메인 화면으로 돌아가게 된다.

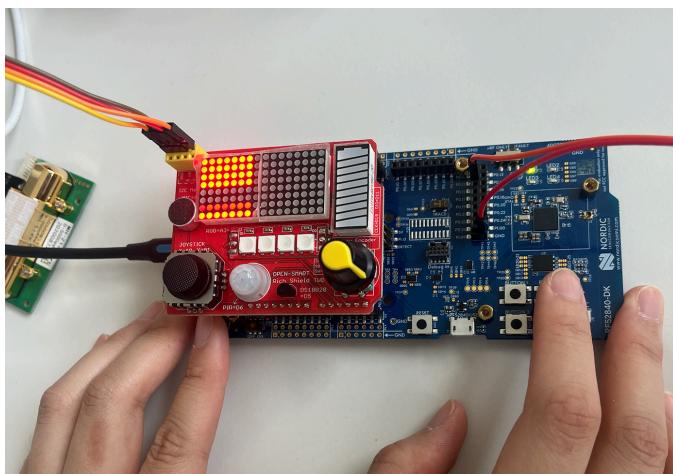


4. CO2값 출력



메인화면에서 버튼 2를 눌러서 CO2측정 모드에 들어왔다면 LED Matrix에 현재 대기중 CO2값이 0~100 사이의 값으로 변환되어어서 시각적으로 표현된다. 좌측에서 우측으로 증가하는 형태로 증가하며, 3초동안 출력한 이후 메인 화면으로 돌아간다.

5. Sound Sensing Mode



버튼 3을 눌러서 Sound Sensing Mode가 시작되었다면 다시 버튼을 누르기 전까지 Sound Sensor로 들어오는 소리의 크기를 CO2와 동일한 모습으로 출력한다. Sound가 감지되지 않으면 X표시가 출력되며, 사용자가 다시한번 버튼 3을 누른다면 메인 화면으로 돌아온다.

3.3 자체 평가

(1) 난이도 평가 (평가 기준에 근거하여 제시)

1. 사용장치의 개수

총 8개의 장치가 사용되었다. 특히 Rich Shield에서 제공하는 장치는 PIR센서와 온도센서를 제외한 모든 장치가 사용되었다.

2. 사용장치의 종류

푸쉬 버튼, LED Matrix, CO2와 같이 수업시간에 실습한 장치들 뿐만 아니라 조이스틱, 로터리 인코더, 소리센서 등과 같이 수업시간에 다루지 않은 장치들도 사용했다. 뿐만 아니라 외부장치인 부저도 사용되었다.

이러한 평가기준으로 평가했을 때 본 프로젝트는 매우 다양하고 많은 장치를 사용했으며, 수업시간에 다루지 않은 까다로운 장치들도 사용했기에 난이도가 높은 프로젝트라고 평가할 수 있다.

(2) 계획대비 완성도

모든 장치가 문제없이 연결되었고 사용자가 사용하기에 무리가 없을 정도로 기능이 완성되었다. 뿐만 아니라 계획했던 기능들 이외에 프로젝트를 진행하면 추가로 필요하다고 판단되는 기능들이 많이 추가되었으며 유저 인터페이스가 직관적이고 깔끔하기 때문에 완성도가 높다고 볼 수 있다.

(3) 조원의 개인 기여 내용

- 이원빈 : BLE, 조이스틱, 로터리 인코더, 배터리 디스플레이, LED Matrix, 보고서 작성
- 이윤서 : 부저, CO2, 소리센서, 어셈블리 작성, 보고서 작성

4. 부록

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

/nrf52840dk_nrf52840.overlay : 디바이스 트리 설정

/prj.conf : 커널 설정

/README.txt : 프로그램 빌드 및 사용 방법

/src

- batteryDisplay.c : 아두이노 rich shield board의 배터리 디스플레이 관련 코드

- ble.c: 핸드폰과의 BLE 통신 관련 코드.

- buzzer.c : nrf52840에 연결하여 사용하는 buzzer 관련 코드

- co2.c : CO2 센서 관련 코드

- gpios.c : gpio 설정 관련 코드

- joy.c : 아두이노 rich shield board의 joystick 관련 코드

- led.c : 아두이노 rich shield board의 led matrix 관련 코드

- main.c : 메인쓰레드

- map.s : Sound Value를 mapping하는 어셈블리 함수

- rotary.c: 아두이노 rich shield board의 rotary encoder 관련 코드.

- sound.c: 아두이노 rich shield board의 sound 관련 코드.

- timer.c : 게임실행시 각 프레임을 정해진 시간 단위로 출력하기 위한 코드

4.2 개발과정에서 에러와 해결 방법

1. PWM 출력값 오류

수동부저를 보드에 정확하게 연결했고, 오버레이 파일에 PWM채널 설정을 정확하게 해주었다. 작성한 코드를 실행시켰을 때 초기화도 정상적으로 진행이 되었으며 같은 PWM채널을 사용하는 LED1이 점등하고 있음에도 부저에서는 아무런 소리가 들리지 않았다. PWM 신호를 보낼 때 pwm_set_dt함수를 사용하게 되는데 해당 함수는 period와 pulse라는 인자를 받는다. 각각의 인자는 아래의 표를 참고해서 입력했다.

```

30  #define sixteenth 38
31  #define eighth 75
32  #define quarter 150
33  #define half 300
34  #define whole 600
35
36
37  #define C4 262
38  #define D4 277
39  #define D4 294
40  #define Eb4 311
41  #define E4 330
42  #define F4 349
43  #define Gb4 370
44  #define G4 392
45  #define Ab4 415
46  #define A4 440
47  #define Bb4 466
48  #define B4 494
49  #define C5 523
50  #define Db5 554
51  #define D5 587
52  #define Eb5 622
53  #define E5 659
54  #define F5 698
55  #define Gb5 740
56  #define G5 784
57  #define Ab5 831
58  #define A5 880
59  #define Bb5 932
60  #define B5 988
61  #define C6 1046
62  #define Db6 1109
63  #define D6 1175
64  #define Eb6 1245
65  #define E6 1319
66  #define F6 1397

```

아무소리가 나지 않았던 이유는 위 사진에 적혀있는 값은 msec단위로 작성된 값인데 zephyr에서 제공하는 pwm_set_dt함수의 인자값은 nano second단위였다. 따라서 기존에 입력한 값은 가청주파수를 벗어나기 때문에 아무런 소리가 들리지 않았던 것이다. 기존 주파수에 10^9값을 곱해줌으로써 해당 문제를 해결하였다.

2. Sound Sensor 연결 실패 문제

```

[00:00:00.251,281] <err> os: ***** USAGE FAULT *****
[00:00:00.251,281] <err> os: Attempt to execute undefined instruction
[00:00:00.251,312] <err> os: r0/a1: 0x00000000 r1/a2: 0x00007afe r2/a3: 0x00000000
[00:00:00.251,312] <err> os: r3/a4: 0x00006ebf r12/ip: 0x00000000 r14/Lr: 0x00000401
[00:00:00.251,342] <err> os: xpsr: 0x61000000
[00:00:00.251,342] <err> os: Faulting instruction address (r15/pc): 0x000000408
[00:00:00.251,373] <err> os: >>> ZEPHYR FATAL ERROR 36: Unknown error on CPU 0
[00:00:00.251,403] <err> os: Current thread: 0x200007f0 (unknown)
[00:00:00.564,025] <err> fatal_error: Resetting system

```

이미 기존의 프로젝트에 joystick이 adc채널을 사용중이었고, sound sensor를 추가해서 channel3으로 설정을 해주었는데 초기화 과정에 위와같은 에러메시지가 출력되었다. 해당문제는 코드 내에서 adc_channels 구조체 배열에 새롭게 추가된 channel3을 추가해주지 않아서 발생한 문제였다.

```

static const struct adc_dt_spec adc_channels[] = {
    ADC_DT_SPEC_GET_BY_IDX(DT_PATH(zephyr_user), 0),
    ADC_DT_SPEC_GET_BY_IDX(DT_PATH(zephyr_user), 1),
    ADC_DT_SPEC_GET_BY_IDX(DT_PATH(zephyr_user), 2),
};

```

3. Assembly 코드를 작성했을 때 flag값 오류

기존에 Sound Sensor에서 사용하던 map함수를 아래와 같은 어셈블리 코드로 바꾸어주었다.

```
map:  
    // int map(int x, int in_min, int in_max, int out_min, int  
    // out_max)  
    // Arguments in registers: x (r0), in_min (r1), in_max (r2),  
    // out_min (r3)  
  
    push r4  
    mov r4, lr  
    push {r4}  
    ldr r4, [sp, #20]  
    sub r5, r0, r1  
    sub r6, r4, r3  
    sub r7, r2, r1  
    mul r5, r5, r6  
    udiv r5, r5, r7  
    add r0, r5, r3  
    pop {r4}  
    mov lr, r4  
    pop r4  
    bx lr
```

이 어셈블리코드를 사용해서 프로그램을 빌드하고 실행했을 때 Sound Sensing값은 정상적으로 출력하지만 Sound Sensing Mode에서 벗어나지 못하는 오류가 발생했다. busy라는 flag를 사용해서 특정 모드가 실행중이라는 여부를 판단하지만 map함수를 실행하고 나니 해당 flag값이 바뀌어 있었다. 해당 오류는 함수 내부에서 사용하는 레지스터 값을 유지하지 않아서 발생하는 오류였다. 어셈블리 코드에서 인자값 4개까지는 r0,r1,r2,r3 레지스터에 저장되어 있지만 그 이후부터는 stack에 저장된다. 따라서 r4레지스터부터는 이전의 프로그램의 context에서 중요한 값들이 담겨있을 수 있다. 따라서 해당 레지스터를 함수 내에서 사용하기 위해서는 그 값을 stack에 저장했다가 복원하는 과정이 필요하다. 따라서 아래의 코드로 수정하였고 문제가 해결되었다.

```
map:  
    // int map(int x, int in_min, int in_max, int out_min, int  
    // out_max)  
    // Arguments in registers: x (r0), in_min (r1), in_max (r2),  
    // out_min (r3)  
  
    push {r4-r7}  
    mov r4, lr  
    push {r4}  
    ldr r4, [sp, #20]  
    sub r5, r0, r1  
    sub r6, r4, r3  
    sub r7, r2, r1  
    mul r5, r5, r6  
    udiv r5, r5, r7  
    add r0, r5, r3  
    pop {r4}  
    mov lr, r4  
    pop {r4-r7}  
    bx lr
```

4.3 개인 소감

이원빈: 학기 중에 배운 마이크로프로세서 관련 지식들을 기말 프로젝트로서 종합하여 녹여내고, 응용할 수 있던 좋은 기회였다고 생각한다. 사실 수업에서 배웠던 내용들은 막연하게 느껴지기도 했고, 생소한 용어들에 다소 어렵게 느껴졌었다. 어셈블리부터 시작해서 nrf52840, zephyr까지 나에게 처음인 모든 것들이 처음엔 어떻게 공부해야 할지, 어떻게 접근해야 할지 몰라서 두려웠다. 하지만 이번 프로젝트를 통해 그러한 개념들을 스스로 다시 공부하고, 정리하여 프로젝트에 직접 녹여내는 과정들을 통해 소프트웨어와 하드웨어의 융합에 대한 자신감을 얻을 수 있었다.

사실, 프로젝트 초반까지도 어려웠다. zephyr의 device tree, prj.conf 등의 역할도 헷갈렸고, 더군다나 overlay에는 무엇들을 어떻게 적어야하는지, 그리고 이미 있는 예제에서는 왜 그렇게 하드웨어적 요소들이 구성되어있는지도 잘 몰랐다. 또한 단순히 논리적 제어만 필요한 것이 아니라 nrf52840과 아두이노 rich shield board, 그리고 zephyr의 융합으로서 하드웨어적 요소까지 고려해야하는 프로젝트이다보니, 관련 변수 하나 잘못 적으면 빌드가 아예 되지 않는 상황에도 답답했다. 하지만 끈기있게 코드들을 보고 다양한 자료들을 찾아보며 하나둘씩 이해하기 시작했고, 프로그래밍을 하며 보드에 센서들이 연결되어 조화로운 데이터들을 주고받음에 차츰차츰 가슴이 벅차올랐다. 프로젝트 중반에는 LED Matrix를 이용하여 게임을 만들며 너무나 즐거워하는 모습에 스스로 놀라기도 했다. 프로젝트 후반에 다양한 센서 모듈들을 이제 융합하는 과정에서 많은 어려움이 있었는데, 그 때 여러 센서들의 충돌을 방지하고자 열심히 디버깅했던 순간이 기억에 남는다. 특히 BLE를 연결하는게 가장 어려웠는데, 기존 프로젝트에 새로운 통신 관련 함수들을 도입하는게 가장 힘들었고, 성공했을 때 가장 뿌듯했다. 보드에서의 데이터가 내 핸드폰에서 보일 때는 정말 짜릿했다.

팀원과는 github를 적극적으로 이용해서 버전 컨트롤을 했고, 각 모듈 별 브랜치를 만들어 main에 merge하고, 충돌을 해결하는 과정을 통해 보다 원활한 프로젝트 관리를 할 수 있었다. git을 통해 협업하는 것이 얼마나 중요하고 유용한지도 새삼 배울 수 있었다. 마지막으로 짧은 기간 안에 완성도 있게 만들어야하는 프로젝트였다보니 기능들의 작동 측면에 많이 집중했던 것 같은데, 코드상으로 아쉬운 부분들도 있다. 하드코딩한 것들도 있고, 비효율적으로 로직을 짜놓은 것도 있어 기회가 된다면 모두 깔끔한 코드로 만들어 유지보수를 용이하게 하고 싶은 마음이 있다.

이윤서: 이번 마이크로프로세서응용 파이널 프로젝트를 진행하면서 하드웨어를 제어하는 것에 대한 막연한 두려움을 극복할 수 있었다. 처음 zephyr와 Rich Shield 보드를 접했을 때, RTOS에 대한 개념을 알지 못했고, 처음 접해보는 센서들을 보드에 부착해 제어한다는 것이 막막하게 느껴졌기 때문에 해당 프로젝트를 진행하는 것에 대한 두려움이 있었다. 하지만 교수님께서 제공해보신 자료들을 하나씩 읽어보고, 예제코드를 실행해보고, 인터넷에서 여러 사람들이 올려둔 자료들과 Zephyr의 공식문서 등을 읽으면서 기능을 하나씩 구현해 나갈 때마다 전체적인 흐름과 구성을 이해하게 되는 것이 느껴졌다. 프로젝트 마지막에는 센서 하나를 더 붙이거나 기능을 추가하는 작업이 전혀 막연하게 느껴지지 않았으며, 어떤 하드웨어가 주어지더라도 그에 대한 API와 하드웨어 정보만 주어진다면 충분히 제어할 수 있을 것이라는 자신감을 갖게 되었다. 이

과정을 통해서 막연하고 어려워보이는 일이라도 부딪혀보고 조급한 마음 없이 조금씩 배워나간다면 생각보다 빠른 시간에 성장할 수 있다는 사실을 경험할 수 있었다.

또한 조원과 함께 협업하면서 Github과 같이 버전 컨트롤 서비스를 활용하는 것이 얼마나 중요한지 알 수 있었고, 파일을 구성할 때 철저한 모듈화와 테스팅을 진행하는 것이 협업에 있어서 서로의 코드에 대한 이해도를 높이는 데에 정말 중요한 역할을 한다는 사실을 알게 되었다.

마지막으로 시스템의 인터럽트를 설정하는 과정에서 인터럽트에서는 오래걸리는 작업을 하기보다는 간단한 작업을 수행하는 것이 더 좋은 코드라는 것과, 시스템에서 제공하는 API를 적극적으로 사용하는 것이 가독성이 좋고 유지보수하기 쉬운 프로그램을 만드는 방법이라는 사실도 알게 되었다. 뿐만 아니라 각 API를 사용할 때 내부적으로 어떤 작업이 이루어지는지를 잘 알고 있어야 여러 작업이 동시에 진행되는 환경에서 각 기능들이 충돌없이 작동할 수 있도록 설계할 수 있다는 것을 배웠다.

4.4 개발과정에서 참조 했던 리소스

- Zephyr Getting Started Guide:

https://docs.zephyrproject.org/latest/develop/getting_started/index.html

- 부저 참고자료 :

<https://devzone.nordicsemi.com/f/nordic-q-a/104525/play-tune-on-piezo-buzzer-using-pwm>

- BLE 참고자료 :

https://docs.zephyrproject.org/latest/samples/bluetooth/peripheral_gatt_write/README.html

<https://docs.zephyrproject.org/latest/samples/bluetooth/peripheral/README.html>

<https://docs.nordicsemi.com/bundle/ncs-latest/page/zephyr/samples/bluetooth/peripheral/README.html>

- ChatGPT