

DMA 기반 SPI 통신 설계

프로젝트 명	DMA + SPI 통신을 이용한 피아노 연주
구성원	21800564 이정언 21900424 안지수 21800017 고강현 21800471 유준호
제출 날짜	2023.06.18

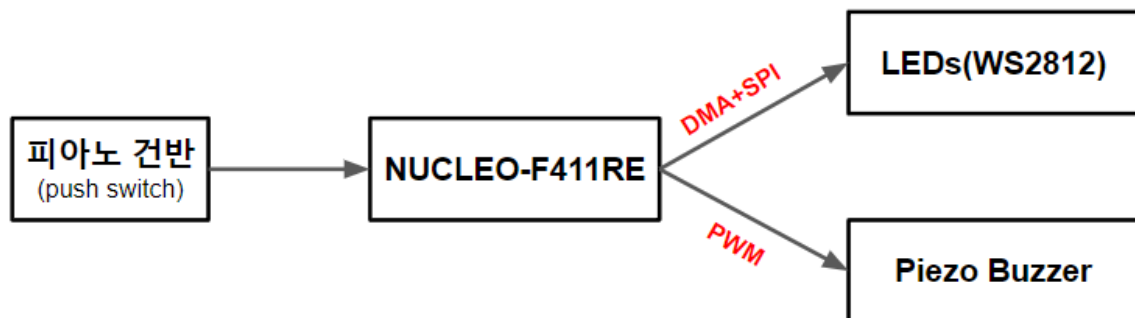
목차

1. 프로젝트 개요	3
1.1 프로젝트 목표	3
1.2 프로젝트 시나리오	3
1.3 프로젝트 제약조건	3
2. 사용된 소자	5
2.1 NUCLEO-F411RE	5
2.2 WS2812 LED bar	7
2.3 푸시 스위치	10
2.4 피에조 부저	10
3. 구현 내용.....	12
3.1 회로 구상 및 구현	12
3.2 PWM 기능 구현	15
3.3 SPI + DMA 기능 구현	18
4. 결과 및 데모	24
5. 결론	27
6. 참조	29

1. 프로젝트 개요

1.1 프로젝트 목표

본 Final Project의 목표는 피아노 건반을 누를 때마다 NUCLEO-F411RE와 WS2812 LED bar간의 DMA 및 SPI 통신을 통해 색상을 제어하고, 피에조 부저로 PWM신호를 송신하여 특정 음계를 출력하는 것이다.



<figure 1.1.1_전체 동작 시스템>

1.2 프로젝트 시나리오

프로젝트의 시나리오는 다음과 같이 총 5단계로 이루어져 있다.

1단계: 피아노 건반용으로 사용하는 스위치 7개 중 1개를 누른다.

2단계: 해당 스위치를 누르면 풀 다운 저항 방식 회로에 전류가 흘러, 연결된 특정 DigitalIn GPIO핀의 전압 값이 '1'의 상태가 되는데, 이때 스위치의 index 값(0~6)을 확인한다.

3단계: Timer 객체를 사용하여 스위치가 눌린 시간을 측정한다.

4단계: 스위치가 눌린 시간에 따라 WS2812 LED bar의 각 pixel의 색상 및 밝기를 제어한다.

5단계: 스위치의 index에 대응하는 음계 주파수 값을 이용하여 주기 값을 설정하여 피에조 부저에서 해당하는 음계가 출력되게 한다.

1.3 프로젝트 제약 조건

본 프로젝트를 진행하는 과정에는 다음과 같은 제약 조건이 주어진다.

1. DMA(Direct Memory Access)와 SPI(Serial Peripheral Interface) 통신을 통해 WS2812 LED bar로 데이터 전송 및 픽셀의 색상 제어를 수행해야 한다.
2. DMA 및 SPI 통신을 수행할 때 관련된 HAL library를 사용하지 않고 register를 직접 제어함으로 구현해야 한다.
3. PWM 신호를 이용하여 피에조 부저의 소리를 출력해야 한다.
4. Mbed-OS의 사용이 가능하다.

2.사용된 소자

본 프로젝트에서 사용한 소자로 NUCLEO-F411RE, WS2812 LED bar, 푸시 스위치, 피에조 부저가 있다.

2.1 NUCLEO-F411RE

1. 기능적 특징

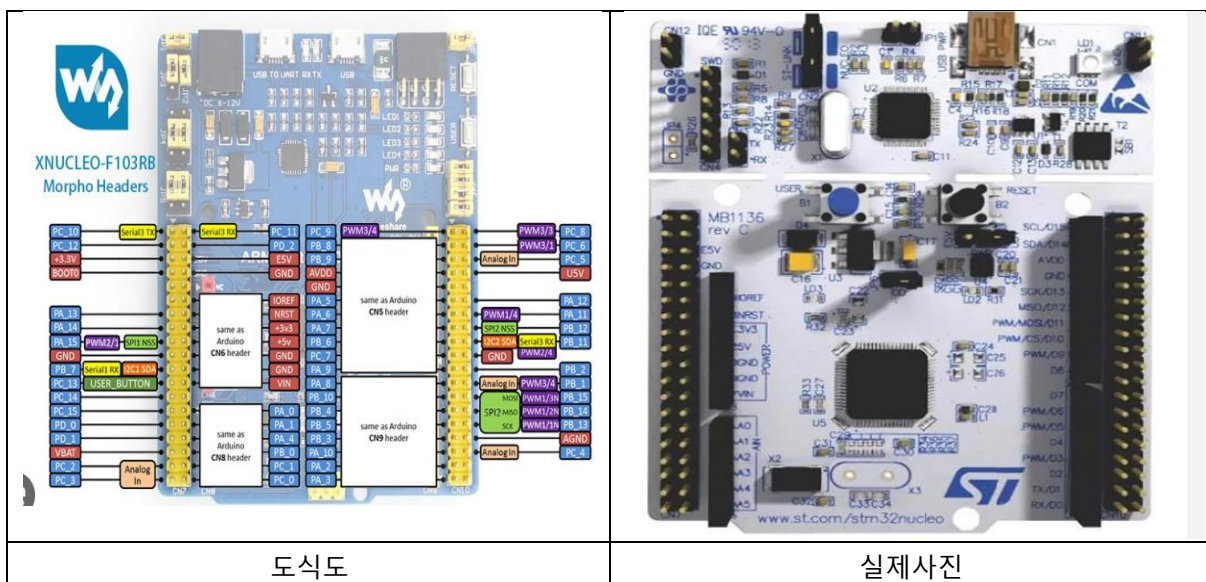
STM32 NUCLEO-F411RE는 ARM Cortex-M4 32비트 RISC 코어를 기반으로 하는 STM32F411RET6 마이크로컨트롤러를 특징으로 하는 개발 보드이다. 이 보드는 사용자가 STM32 마이크로 컨트롤러로 새로운 개념을 시도하고 프로토타입을 구축할 수 있는 유연한 방법을 제공하도록 설계되었다.

NUCLEO-F411RE 보드의 주요 기능은 다음과 같다.

- **Microcontroller:** LQFP64 패키지의 STM32F411RET6
- ARM 32비트 Cortex-M4 CPU(FPU 포함)
- 100MHz 최대 CPU 주파수
- 1.7V ~ 3.6V의 VDD
- 512KB 플래시
- 128KB SRAM
- 외부 인터럽트 기능이 있는 GPIO(50)
- 12비트 ADC, DAC, I2C, UART, SPI, SDIO 등
- 보드 전원 공급 장치: USB 버스를 통해 공급받거나 또는 외부에 3.3V 또는 5V 공급이 가능하다.
- 외부 애플리케이션 전원 공급 장치: 3V, 5V
- On-board debugger: SWD 커넥터가 있는 ST-LINK/V2-1 디버거/프로그래머. 또한 키트를 독립형 ST-LINK/V2-1로 사용하기 위한 별도의 선택 모드 스위치가 있다.
- User LED: 녹색 LED인 LD2는 디버거 없이 출력을 테스트하거나 프로그램의 특정 상태를 나타내는 데 도움이 된다.

- 사용자 및 재설정 버튼
- 보드 커넥터: Arduino™ Uno V3 확장 커넥터
- 모든 STM32 I/O에 대한 전체 액세스를 위한 STMicroelectronics Morpho 확장 핀 헤더
- **IAR™, Keil® 및 GCC 기반 IDE를 포함하여 **광범위한 통합 개발 환경(IDE)**을 지원한다.
- 확장 커넥터: Nucleo 보드는 Arduino Uno Rev3 연결 및 ST 모르포 확장 핀 헤더의 두 가지 유형의 확장 리소스를 제공한다.
- 플렉서블 보드 전원 공급 장치: 보드는 USB 케이블로 전원을 공급받지만 외부 소스로도 전원을 공급받을 수 있다.
- 다양한 패키지 소프트웨어 예제와 함께 포괄적인 STM32 소프트웨어 HAL 라이브러리가 있다.

이 보드는 프로토타입을 쉽게 만들 수 있도록 설계되었으며 다양한 전원 공급 장치 및 I/O에 유연하다는 특징을 가진다. 또한 Arduino와의 호환성으로 Arduino 생태계에 익숙한 사람들도 쉽게 사용할 수 있다.



<Figure2.1.1_NUCELO-F411RE>

2. 사용되는 Development Tool

통합 개발 환경(IDE): 소프트웨어 개발을 위해 컴퓨터 프로그래머에게 포괄적인 기능을 제공하는 소프트웨어 응용 프로그램이다.

- STM32CubeIDE: STMicroelectronics의 STM32용 무료 Eclipse 기반 IDE입니다. 여기에는 STM32Cube MCU 패키지, 임베디드 소프트웨어 라이브러리 및 미들웨어 스택이 포함된다.
- Keil uVision MDK: 업계에서 널리 사용되는 Cortex-M 기반 MCU용 IDE이다.
- IAR Embedded Workbench for Arm: 이것은 무료는 아니지만 또 다른 유명한 Arm 개발용 IDE이다.
- System Workbench for STM32(SW4STM32): STM32 개발에 사용할 수 있는 무료 Eclipse 기반 IDE이다.
- Atollic TrueSTUDIO: 이것은 STM32 개발에 유용한 많은 기능을 갖춘 Eclipse 기반의 상업적으로 향상된 C/C++ IDE이다.
- Arduino IDE: 더 간단한 프로젝트의 경우 또는 Arduino 실드 또는 라이브러리를 사용하는 경우 옵션이 될 수 있다.

프로그래밍/디버깅 도구: NUCLEO-F411RE 보드에는 SWD(Serial Wire Debug) 프로토콜을 지원하는 통합 ST-LINK/V2-1 디버거 및 프로그래머가 함께 제공됩니다. STM32 마이크로컨트롤러를 프로그래밍하고 애플리케이션을 디버그하는 데 사용할 수 있다.

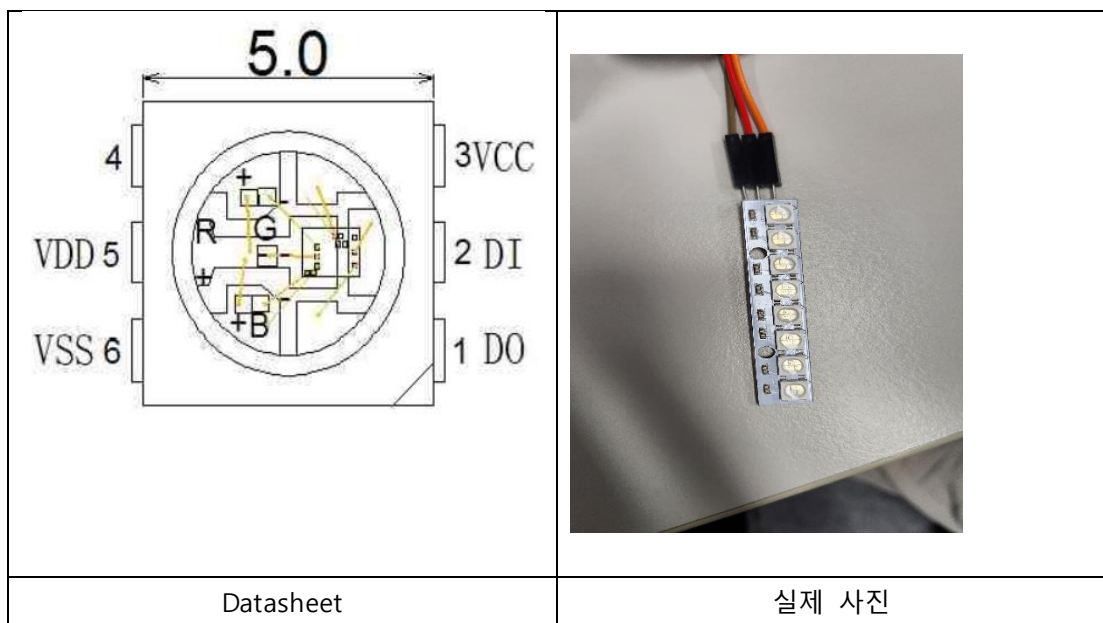
- 소프트웨어 라이브러리 및 미들웨어: STM32F4 시리즈용 STM32Cube MCU 패키지에는 HAL(Hardware Abstraction Layer), LL(Low Layer) API 및 미들웨어 스택이 포함되어 있다. 또한 응용 프로그램을 개발할 때 유용할 수 있는 많은 예제를 제공한다.
- RTOS: 복잡한 애플리케이션을 개발하는 경우 실시간 운영 체제(RTOS) 사용을 고려할 수 있다. FreeRTOS는 STM32용으로 널리 사용되며 STM32Cube에 포함되어 있다.

이러한 도구는 전문 IDE 및 HAL 라이브러리를 사용하는 저수준 개발부터 Arduino IDE 및 라이브러리를 사용하는 고수준 개발에 이르기까지 STM32 NUCLEO-F411RE 보드용 소프트웨어 개발을 위한 광범위한 옵션을 제공한다.

2.2 WS2812B LED

1. 구조적 특징

WS2812는 LED 소자로, 내부에는 Red, Green, Blue LED가 있다. 이 LED들은 각각의 밝기를 조절하여 원하는 색을 표현할 수 있다. 따라서 WS2812를 사용하면 다양한 색상을 표현할 수 있다. 또한, WS2812는 Neopixel이라고도 불리며, 각각의 LED에 대한 개별 제어(색상 및 ON/OFF)가 가능하다. 이는 각 LED마다 원하는 색을 설정하거나 특정 LED를 켜거나 끌 수 있다는 것을 의미한다. 또한, WS2812는 배선 연결이 간단하여 원하는 모양으로 서로 연결하여 사용할 수 있다.



<figure2.2.1_WS2812 LED>

NO.	Symbol	Function description
1	DOUT	Control data signal output
2	DIN	Control data signal input
3	VCC	Power supply control circuit
4	NC	
5	VDD	Power supply LED
6	VSS	Ground

<figure2.2.2_Table. Pin matching of WS2812>

2. 제어적 특징

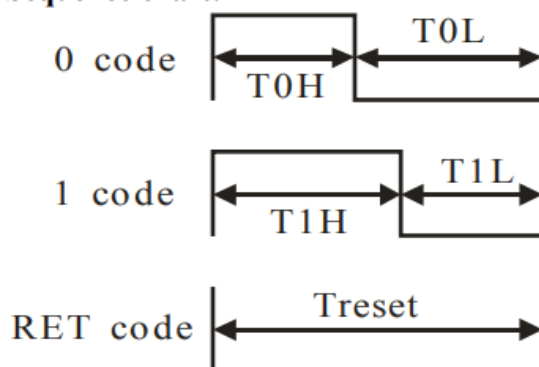
WS2812를 제어하기 위해서는 해당 LED에 원하는 색을 표현하기 위한 신호를 입력해 주어야 한다. 이 신호는 DIN(Digital Input)으로 입력되며, 하나의 LED 당 24bit의 데이터로 구성된다. 이 24bit 데이터는 8bit씩 G/R/B의 순서대로 색상에 대한 데이터를 전달하게 된다. 따라서 각 색상의 밝기는 256단계로 표현할 수 있다. 예를 들어, R(빨강)에 대한 데이터가 0이면 해당 LED는 꺼지고, 255이면 최대 밝기로 켜진다. 이렇게 각각의 LED에 대한 개별적인 색상과 ON/OFF 제어가 가능하다.

WS2812에서 인식하는 1bit의 '0', '1'값을 전송하기 위해서는 figure 2.2.3를 참고하여 데이터 전송시간을 조절해줘야 한다. '0'으로 인식하기 위해서는 HIGH전압으로 0.35us, LOW전압으로 0.8us를 보내야 한다. 한편, '1'의 경우는 HIGH전압으로 0.7us, LOW전압으로 0.6us만큼 전송 시간을 설정해야 한다. 정리하자면, 신호 주기는 1.25us(주파수 800kHz)로 동일 하지만 전압이 HIGH, LOW인 시간을 다르게 조절하여 '0' 혹은 '1'의 1bit 값을 구성할 수 있다.

Data transfer time($T_H+T_L=1.25\mu s \pm 600ns$)

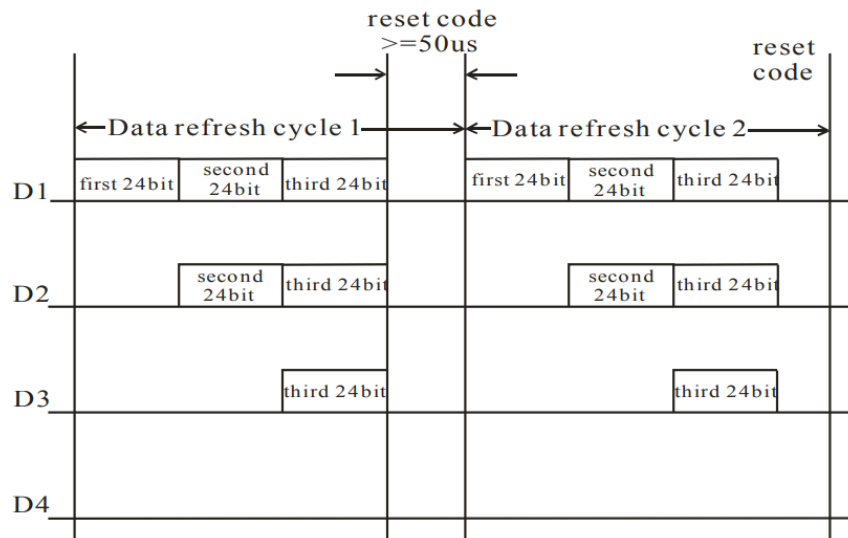
T0H	0 code ,high voltage time	0.35us	$\pm 150ns$
T1H	1 code ,high voltage time	0.7us	$\pm 150ns$
T0L	0 code , low voltage time	0.8us	$\pm 150ns$
T1L	1 code ,low voltage time	0.6us	$\pm 150ns$
RES	low voltage time	Above 50 μs	

Sequence chart:



<figure2.2.3_ Control feature of WS2812>

Data transmission method:



Note: The data of D1 is send by MCU, and D2, D3, D4 through pixel internal reshaping amplification to transmit.

Composition of 24bit data:

G7	G6	G5	G4	G3	G2	G1	G0	R7	R6	R5	R4	R3	R2	R1	R0	B7	B6	B5	B4	B3	B2	B1	B0
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

Note: Follow the order of GRB to sent data and the high bit sent at first.

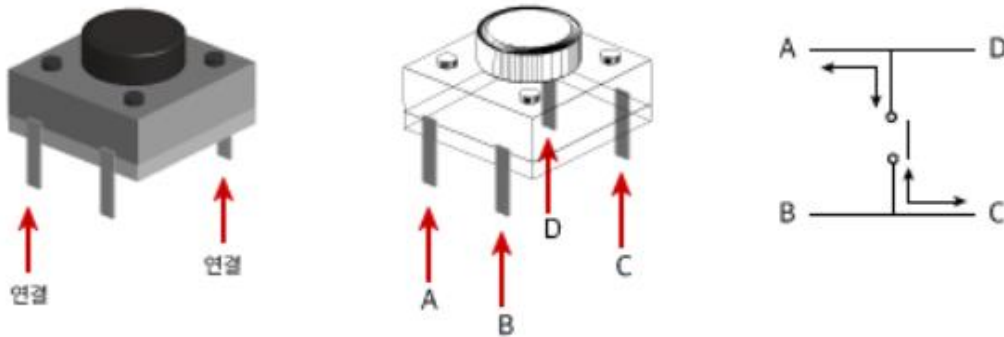
<figure2.2.4_Transmission method of WS2812>

본 프로젝트에서는 8개의 LED가 일렬로 배열된 WS2812 LED스트립을 사용하여 8비트 이진 형식으로 표현했다. LED는 NUCLEO보드에 연결되었으며, 3.3V로 전원이 공급되고 GND에 연결하였다.

2.3 푸쉬 스위치

스위치란 전자제품의 전원을 on/off시키거나 전자기기 전류의 흐름을 제어할 수 있는 장치다. 푸시 버튼 스위치는 상단의 단추 모양 버튼을 누름으로 figure 2.3.1의 A와 C 그리고 B와 D간에 전류가 흐르도록 연결시켜 준다. 주의할 점은 A와 D, B와 C는 각각 연결되어 있어 버튼을 누르

지 않더라도 전류가 흐른다는 것이다.



<figure2.3.1_스위치 구성>

2.4 피에조 부저

피에조 부저는 피에조 효과 혹은 압전 효과 성질을 이용하여 소리를 내는 장치이다. 압전 효과란 어떤 물질에 압력을 가했을 때 전기적인 변화가 생기는 현상이다. 피에조 부저는 이러한 원리로 압전 물질에 전압을 인가하여 이를 응축 혹은 신장시키고, 여기에 얇은 판을 붙여주어 인가되는 전압에 따라 얇은 판을 떨리게 하여 소리를 낸다.

피에조 부저의 종류로는 능동형 부저(active buzzer), 수동형 부저(passive buzzer)가 있고, 각각의 특징은 다음과 같다.

능동형 부저(active buzzer)	수동형 부저(passive buzzer)
1. 회로가 내장되어 있음 ➔ 전원 인가 시 바로 소리 발생 2. 사용하기에 편리 3. 낼 수 있는 소리가 한 가지로 고정	1. 다양한 음역대의 소리 재생 가능 2. 주파수를 이용한 프로그램 작성 필요

본 프로젝트에서 사용하는 수동형 부저는 figure 2.4.1로 PWM신호로 주파수를 변경하여 원하는 음역대의 소리를 재생한다.

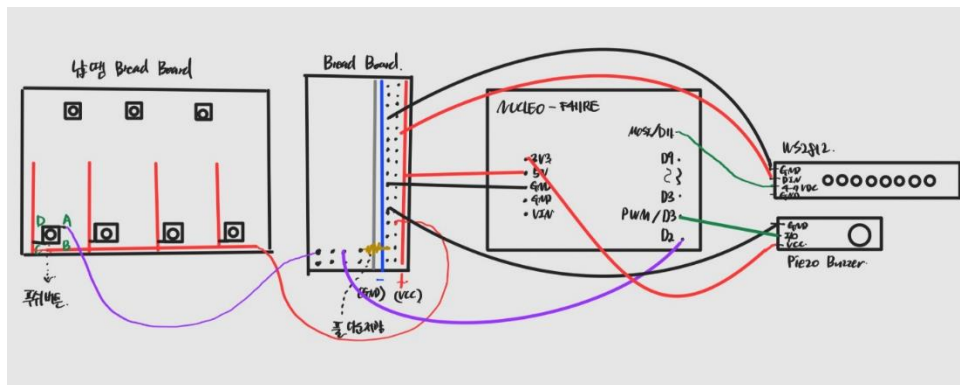


<figure2.4.1_수동형 피에조 부저>

3.구현 내용

3.1 회로 구상 및 구현

임베디드를 이용한 피아노 프로젝트를 하기위한 전체적인 회로 구상도이다.



<figure3.1.1_임베디드 피아노 회로구상도>

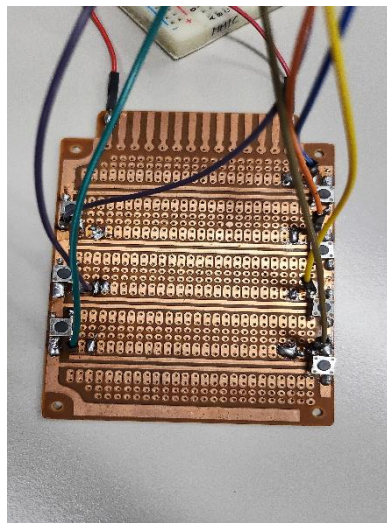
피아노 건반 7개를 구현하기 위해서 푸쉬 버튼 총 7개를 납땜할 수 있는 bread board에 연결하고 각각 D2, D4, D5, D6, D7, D8, D9 pin에 연결하였으나 회로 구상도의 간략화를 위해 첫번째 푸쉬 버튼과 D2 pin과의 연결만 표현하였다.

<회로pin 연결>

WS2812 LED	DIN -> VCC(5V) 4-7VDC -> PWM/MOSI/D11 GND -> GND
Piezo Buzzer	GND -> GND I/O -> D3(PB_3) VCC -> VCC(3.3V)
푸쉬 스위치	A-D -> GND B-C -> VCC(5V)

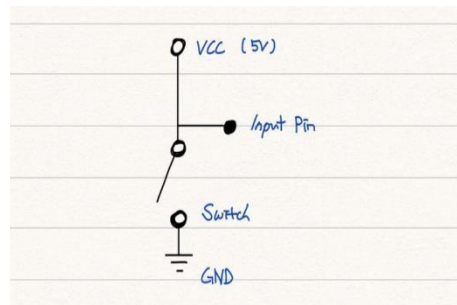
풀다운 저항	위 -> VCC(5V) 아래 -> GND
--------	---------------------------

피아노 건반을 만들기 위해서 푸시 스위치를 사용한 모습이다. 푸시 스위치의 한쪽에는 VCC(5V)를 연결하여 전압을 가했다. 스위치의 다른 쪽에는 전선으로 bread board에 연결하였다.



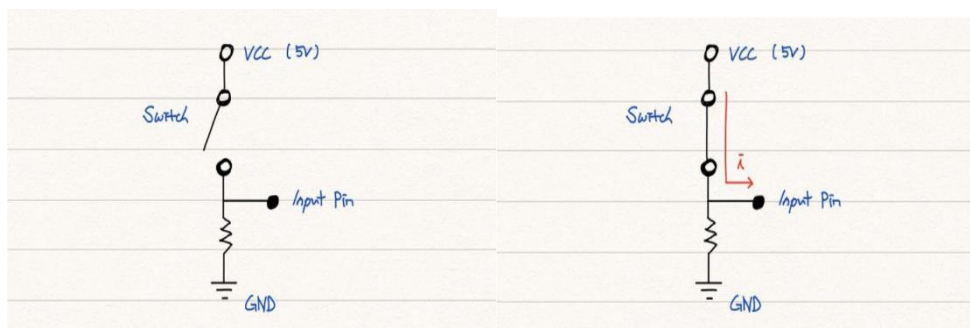
<figure3.1.2_납땀을 할 수 있는 breadboard에 버튼 스위치와 전선을 납땀한 모습>

Breadboard에서 버튼 스위치에서 나온 전선을 아래 그림과 같이 VCC(5V)에 연결을 하였다. 또한 그 밑에는 GPIO pin(D2~D9, D3는 제외)을 연결하였다. 아래 부분에 풀 다운 저항을 달아 최종적으로는 GND에 전선이 연결되게끔 회로를 구성하였다. 중간에 풀 다운 저항을 연결한 이유는 다음과 같다.



<figure3.1.2_플로팅 현상>

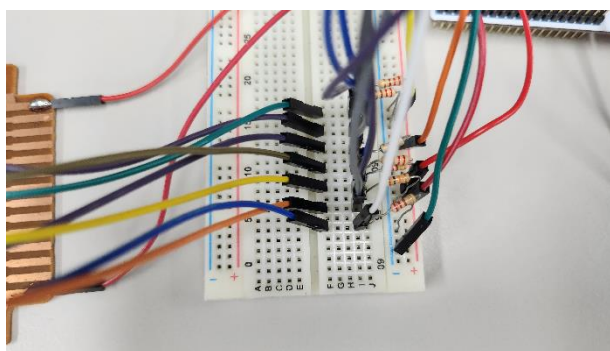
스위치가 연결되면 전류가 정상적으로 흐르게 된다. 스위치가 연결되지 않은 상태에서 전류가 흐르는지 안 흐르는지 알 수 없는 상태가 있는데 이를 플로팅 현상이라 한다. 디지털 핀을 입력상태로 설정하면 5V값은 HIGH 0V값은 LOW 로 인식한다. 그런데 핀 자체의 칩에서는 0V, 5V인지 인식할 수 가 없어서 플로팅 현상이 생기게 된다. 즉 입력단자 주위에 정전기나 잡음에 의해서 오류가 생기게 된다. 이를 방지하기 위해 풀 다운 저항을 회로에 연결해줬다.



<figure3.1.3_풀 다운저항 스위치 열린 상태(좌), 스위치 닫힌 상태(우)>

풀 다운 저항은 위사진과 같이 밑에 저항을 연결하는 방식이다. 스위치가 열린 상태에서는 어디에도 전류가 흐르지 않고 입력 핀에는 0V 전압이 걸리게 된다.

다음은 스위치가 닫힌 상태이다. GND쪽에 저항이 연결되어 있다. 밑의 저항으로 인해 전류는 모두 입력 핀 쪽으로 흐르고 입력 핀에는 전원 전압과 같은 5V가 걸리게 된다.



<figure3.1.4_실제 풀 다운 저항을 bread board에 연결>

NUCLEO-F411RE의 GND와 VCC(5V)를 각각 bread board의 GND(파란선 -), VCC(빨간선 +)에 연결하였다.



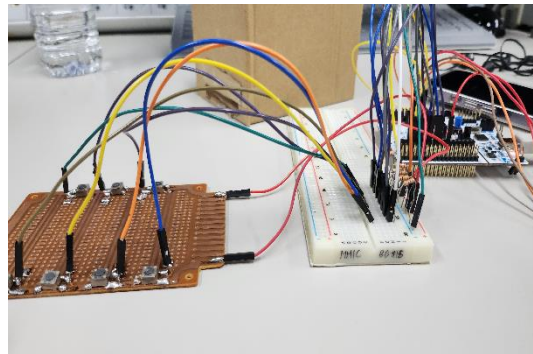
<figure3.1.5 WS2812 LED에 연결할 수 있는 pin>

위 사진을 보면 WS2812에 연결할 수 있는 pin이 GND, DIN, 4-7VDC 총 3가지가 있다. GND에는 breadboard의 GND에 연결, DIN에는 breadboard의 DIN에 연결하였고, 4-7VDC pin에는 NUCLEO-411RE 보드에 있는 PWM/MOSI/D11 pin을 연결하여 DMA기반 SPI방식으로 WS2812 LED를 제어할 수 있도록 회로를 구성했다.



<figure3.1.6 Piezo Buzzer 에 연결할수 있는 pin>

앞서 제시한 Piezo Buzzer의 사진과 같이 Piezo Buzzer에는 GND, I/O, VCC 총 3가지 연결할 수 있는 PIN이 존재하는데 각각 breadboard의 GND, NUCLEO-411RE 보드에 있는 D3(PB_3) pin, VCC(3.3V)에 연결하였다. I/O핀을 D3핀에 연결하여 PWM방식으로 각 스위치마다 할당된 주파수를 Piezo Buzzer로 보내 피아노 건반을 누를 때마다 해당 음을 소리내도록 회로를 구성하였다.



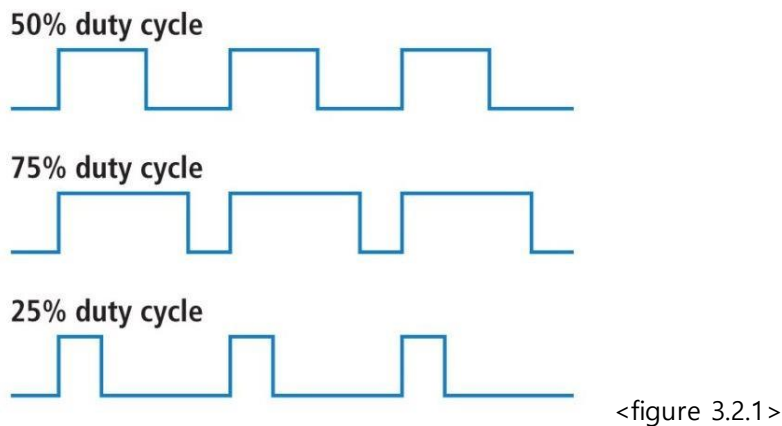
<figure3.1.7 임베디드 피아노 회로 실제사진>

위 사진은 최종적으로 구현된 임베디드를 이용한 피아노 회로 사진이다.

3.2 PWM 기능 구현

앞선 3.1의 회로 구성대로 뉴클레오 보드의 VCC핀을 통해 피에조 부저에 전원을 인가하고 D3핀으로 PWM신호를 보내서 소리를 발생시키고자 한다. 또한, 각각의 스위치를 눌렀을 때 피에조 부저에서 해당하는 음계가 발생하도록 구현하고자 한다. 이를 위해서 사용하는 PWM신호와 음계별 주파수 표를 이해해야 한다.

PWM신호는 pulse width modulation의 약자로 디지털 신호 중 특정한 형태를 띤 신호를 일컫는 용어이다. 이 신호는 디지털 신호의 출력이 HIGH인 시간과 LOW인 시간의 비율, duty cycle을 조정해서 아날로그 신호의 효과를 낸다. Duty cycle 값이 증가할수록 HIGH인 시간이 늘어나고, 이를 통해 피에조 부저에 전원이 인가되므로 소리의 크기를 높일 수 있다. 또한, 주기 값을 조절하여 원하는 음계를 출력할 수 있다. 참고하는 음계별 주파수 표는 figure3.2.2와 같다.



옥타브 및 음계별 표준 주파수

(단위 : Hz)

옥타브 음계	1	2	3	4	5	6	7	8
C(도)	32.7032	65.4064	130.8128	261.6256	523.2511	1046.502	2093.005	4186.009
C#	34.6478	69.2957	138.5913	277.1826	554.3653	1108.731	2217.461	4434.922
D(레)	36.7081	73.4162	146.8324	293.6648	587.3295	1174.659	2349.318	4698.636
D#	38.8909	77.7817	155.5635	311.1270	622.2540	1244.508	2489.016	4978.032
E(미)	41.2034	82.4069	164.8138	329.6276	659.2551	1318.510	2637.020	5274.041
F(파)	43.6535	87.3071	174.6141	349.2282	698.4565	1396.913	2793.826	5587.652
F#	46.2493	92.4986	184.9972	369.9944	739.9888	1479.978	2959.955	5919.911
G(솔)	48.9994	97.9989	195.9977	391.9954	783.9909	1567.982	3135.963	6271.927
G#	51.9130	103.8262	207.6523	415.3047	830.6094	1661.219	3322.438	6644.875
A(라)	55.0000	110.0000	220.0000	440.0000	880.0000	1760.000	3520.000	7040.000
A#	58.2705	116.5409	233.0819	466.1638	932.3275	1864.655	3729.310	7458.620
B(시)	61.7354	123.4708	246.9417	493.8833	987.7666	1975.533	3951.066	7902.133

<figure 3.2.2>

스위치가 눌릴 때마다 PWM신호를 사용하여 피에조 부저에서 해당하는 음계를 출력하도록 구현한 코드는 아래와 같다.

```
PwmOut buzzer(PB_3);
float freq[] = {261.63,293.66,329.63,349.23,392.00,440.00,493.88, 523.25};
DigitalIn switchInputs[] = {D2, D4, D5, D6, D7, D8, D9}; // Array of switch
char buf[MAXIMUM_BUFFER_SIZE];
UnbufferedSerial pc(CONSOLE_TX, CONSOLE_RX, 115200);
```

- ~보드의 PB_3핀으로 PWM신호를 출력하는 buzzer 객체 생성
- 4옥타브 도, 레, 미, 파, 솔, 라, 시, 도에 해당하는 주파수 값을 가진 배열 선언
- ~보드의 D2~D9 핀들을 DigitalIn으로 설정하여 인가되는 전압 값 확인
- 어느 스위치에 전압이 걸리는지 "TeraTerm" 프로그램으로 확인하기 위해 unbufferedserial 객체 생성

```
float period_us; //PWM신호의 주기
int on_idx;

while (1) {
    on_idx=-1;

    //스위치 배열에서 ON된 것 확인
    for(int i=0; i<sizeof(switchInputs)/sizeof(switchInputs[0]); i++){
        if((float)switchInputs[i].read()>0.95f){
            on_idx=i;
            break;
        }
    }
}
```

- 7개의 스위치 중 1개의 스위치가 눌렸을 때 digitalin 핀으로 인가되는 전압 값이 '1'인 경우를 확인하여 해당하는 스위치 번호를 on_idx 변수에 저장

```
sprintf(buf,"눌린 스위치 번호: %d\r\n", on_idx); /
pc.write(buf,strlen(buf));

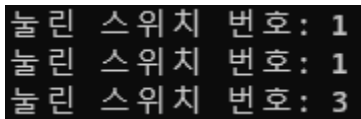
//부저 소리 재생
buzzer = 0.9; //duty cycle 90%를 의미
period_us = 1000000/freq[on_idx]; //주기 T = 1s/fr
buzzer.period_us(period_us);
ThisThread::sleep_for(chrono::milliseconds(100));

//아무 스위치도 눌리지 않았을 때
if(on_idx==-1) buzzer=0;
}
```

- Buzzer 객체에 0.9 값을 대입함으로 duty cycle을 설정할 수 있다. 이는 HIGH 시간이 주기에서 90%를 차지하는 것을 의미하여 보다 큰 소리를 낼 수 있다.

- 주기에 해당하는 변수 period_us에는 $1/\text{frequency} = T(\text{주기})$ 성질을 이용하여 특정 음계 값을 피에조 부저가 출력하도록 설정한다.

결과적으로, 1번 스위치를 누르고 3번 스위치를 눌렀을 때 "TeraTerm" 프로그램에서 figure3.2.3과 같은 결과가 출력됨을 확인할 수 있고 피에조 부저에서 해당하는 음계 '레', '파'가 출력됨을 확인할 수 있었다.



<figure 3.2.3>

3.3 SPI + DMA 기능 구현

1. SPI 작동 방식

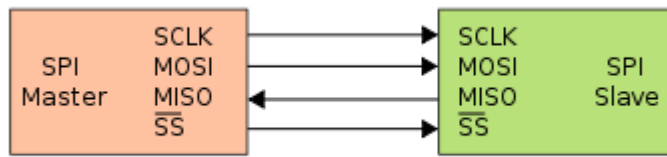
SPI 통신은 Master와 Slave 사이에서 통신하는 방식이다. SPI에서는 송신 data line과 수신 data line이 분리되어있기 때문에 전 이중방식(full duplex) 방식으로 통신이 가능하다. 그러므로 일반적인 I2C 통신보다 빠른 속도의 data 전송이 가능하다. 일반적으로 Master와 Slave 사이의 통신은 4개의 신호 선을 사용한다. 각 신호 선에 대한 설명은 다음과 같다.

SCLK(Serial Clock) : Master 장치에서 clock 신호를 생성하고 전송 속도를 조절한다. 이 clock신호는 데이터 전송의 타이밍을 제어한다.

MOSI(Master Output, Slave Input) : 마스터 장치에서 데이터를 전송하는 신호선이다. 마스터는 MOSI 신호선을 통해 데이터를 슬레이브 장치로 보낸다.

MISO(Master Input, Slave Output) : 슬레이브 장치에서 마스터로 데이터를 전송하는 신호선이다. 마스터는 MISO 신호선을 통해 슬레이브 장치로부터 데이터를 수신한다.

/SS(Slave Select) : 마스터 장치에서 특정 슬레이브 장치를 선택하는 신호선이다. 마스터는 SS 신호선을 통해 특정 슬레이브를 활성화하여 통신한다.



<figure 3.3.1>

위의 figure 3.3.1과 같이 Master와 Slave사이에서 데이터의 송수신이 이루어진다. 이 때 SCLK의 극성과 데이터 전송 규칙에 대해 SPI에는 4가지의 MODE가 있다. 4가지의 MODE는 아래 table #와 같다.

MODE	CPOL	CPHA	설명	사진
0	0	0	클럭의 휴식 상태는 로우 레벨이며, 데이터는 상승 에지에서 샘플링된다.	SPI_MODE0
1	0	1	클럭의 휴식 상태는 로우 레벨이며, 데이터는 하강 에지에서 샘플링된다.	SPI_MODE1
2	1	0	클럭의 휴식 상태는 하이 레벨이며, 데이터는 하강 에지에서 샘플링된다.	SPI_MODE2
3	1	1	클럭의 휴식 상태는 하이 레벨이며, 데이터는 상승 에지에서 샘플링된다.	SPI_MODE3

<table #>

데이터 전송 시점은 데이터 비트가 실제로 전송되거나 샘플링되는 시점을 의미한다. MODE가 1 또는 2인 경우, 데이터는 클럭의 하강 에지에서 전송 또는 샘플링된다. MODE가 0 또는 3인 경우, 데이터는 클럭의 상승 에지에서 전송 또는 샘플링된다. 데이터 전송 시점은 데이터가 올바르게 해석되고 동기화되는지 확인하는 데 중요한 역할을 하게된다. SPI MODE 선택은 마스터 장치와 슬레이브 장치 간의 상호 작용에 대한 규칙을 정의한다. 따라서 마스터와 슬레이브 장치가 동일한 모드로 구성되어야 데이터를 정확하게 전송할 수 있다.

2. SPI통신을 이용한 WS2812 제어

본 실험에서 SPI는 WS2812 LED와 NUCLEO board 간에 통신을 하는데 사용되었다. NUCLEO board가 MASTER, WS2812 LED는 SLAVE 기능을 하였다. NUCLEO board에서는 8개의 LED 각각에 해당하는 R, G, B값을 WS2812로 보내줘야 한다. R, G, B 각각은 색을 표현하기 위해 0~255의 값을 필요로 한다. 때문에 보통의 경우라면 NUCLEO board에서는 WS2812로 24바이트의 RGB값을 보내주면 된다. 하지만 WS2812는 위에서 다뤘듯이 특이

한 타이밍 구조를 가진다. WS2812의 관점에서 1은 0b11111000, 0은 0b11000000 의 값을 가지는 1바이트가 타이밍 조건에 들어왔을 때 인식하게 된다. 타이밍 조건을 만족시키기 위해 SPI의 Baud Rate bits를 011로 설정하여 $f_{PCLK}/16$, 즉 800kHz의 전송 속도를 만족하였다.

	SPI Control Register	내용
Bit 11	DFF(Data frame format)	송신 format을 8 bit 또는 16 bit로 설정한다.
Bit 9	SSM(Software slave management)	SPI 통신에서 슬레이브 장치의 선택과 해제를 소프트웨어적으로 관리한다.
Bit 7	LSBITFIRST(Frame format)	SPI 통신에서 데이터 전송 시 비트 순서를 지정한다.
Bit 6	SPE(SPI enable)	SPI 통신을 활성화하는 설정이다.
Bits 5:3	BR(Baud rate control)	SPI 통신의 전송 속도를 지정한다.
Bit 2	MSTR(Master selection)	SPI 통신에서 마스터 또는 슬레이브 역할을 선택한다.
Bit 1	CPOL(Clock polarity)	SPI Clock 휴식 상태를 지정한다.
Bit 0	CPHA(Clock phase)	데이터 전송 시 클럭의 상승/하강 에지에 대한 설정이다.

	RCC(APB2ENR) Register	내용
Bit 12	SPI1EN(SPI1 Enable)	SPI1(Serial Peripheral Interface 1)의 클럭 활성화 또는 비활성화를 설정.

<table 3.3.2>

```
RCC->APB2ENR |= RCC_APB2ENR_SPI1EN;
SPI1->CR1 |= (0b1U << 9);
SPI1->CR1 |= (0b1U << 6);
SPI1->CR1 |= (0b011U << 3);
SPI1->CR1 |= (0b1U << 2);
SPI1->CR1 |= 0b1U;
```

<figure 3.3.2>

3. DMA를 이용한 SPI통신

Direct Memory Access(이하 DMA)는 보내는 곳의 주소에서 목적지 주소로 직접 data전송하는 것을 의미한다. 즉 CPU를 거치지 않고 데이터의 송신이 이루어진다. CPU의 간섭이 없기에 빠른 data 송신을 보장할 수 있다. 이 때 설정에 따라 Memory-to-Memory, Memory-to-Peripheral, Peripheral-to-Memory 3가지 방향으로 데이터를 보낼 수 있다. Nucleo-F411RE에는 2개의 DMA가 있다. 각 8개의 Stream이 있으며, Stream에는 8개의 Channel이 존재한다. 즉 64가지의 경우의 수 중에서 데이터 송신 환경에 맞추어 선택하여 사용하면 된다. 각 Stream에서 데이터의 송신이 이루어지며, 해당하는 Stream의 Interrupt를 감지하여 송신상태를 확인할 수 있다. 또한, DMA를 사용하여 데이터를 전송하려면 먼저 10개의 DMA register에 대해 초기 설정을 해야한다.

본 프로젝트에서 DMA가 사용된 목적은 사용자가 미리 설정해둔 RGB값을 CPU를 거치지 않고 WS2812로 송신하는 데에 있다. 그러므로 데이터 전송 방향을 Memory-to-Peripheral로 선택하고, Peripheral Address(PAR)를 SPI1->DR의 주소값으로 설정했다. 또한, DMA2의 request mapping을 참고하여 SPI1_TX을 사용하기 위해 channel3과 stream3을 사용했다. 추가적으로 Tx buffer DMA를 활성화하기 위해 SPI Control Register2의 TXDMAEN bit를 사용하였다. 이때 사용된 DMA register에 대한 설명은 table 3.3.3에 나와 있고, 구체적인 설정은 figure 3.3.3에 나와있다.

	DMA Stream x Configuration Register	
Bits 27:25	CHSEL(Channel selection)	데이터 전송에 사용할 특정 채널을 선택한다.
Bits 17:16	PL(Priority level)	DMA 컨트롤러에서 여러 개의 DMA 요청이 동시에 발생할 때, 우선순위를 지정한다.
Bit 10	MINC(Memory increment mode)	DMA 컨트롤러가 데이터 전송 동안 메모리 주소 포인터의 동작을 제어한다.
Bit 8	CIRC(Circular mode)	Tx 버퍼의 값을 모두 보내면 다시 처음 주소로 돌아갈지 결정한다.
Bits 7:6	DIR(Data transfer direction)	DMA 전송의 데이터 전송 방향을 지정하는 설정이다.
Bit 4	TCIE(Transfer complete interrupt enable)	DMA 전송이 완료되면 인터

		럽트를 발생시켜 해당 이벤트를 처리할 수 있다
Bit 0	EN(Stream enable)	DMA 스트림을 활성화하는 설정이다.

	DMA Stream x Peripheral Address Register	내용
Bit 31:0	PAR[31:0] (Peripheral address)	DMA 채널과 연결된 주변 장치의 주소를 설정하는 레지스터이다.

	RCC(APB1ENR) Register	내용
Bit 22	DMA2EN (DMA2 Enable)	DMA2 컨트롤러의 클럭 활성화 또는 비활성화를 설정하는 데 사용된다.

	SPI Control Register 2	내용
Bit 1	TXDMAEN (TX buffer DMA Enable)	데이터 전송 시 전송 버퍼의 DMA 전송을 활성화하는 데 사용된다.

<table 3.3.3>

```

SPI1->CR2 |= (0b1U << 1);

RCC->AHB1ENR |= RCC_AHB1ENR_DMA2EN;

DMA2_Stream3->CR |= (0b011U << 25);
DMA2_Stream3->CR |= (0b11U << 16);
DMA2_Stream3->CR |= (0b1U << 10);
DMA2_Stream3->CR |= (0b1 << 8);
DMA2_Stream3->CR |= (0b01U << 6);
DMA2_Stream3->CR |= (0b1U << 4);
DMA2_Stream3->CR |= (0b0U << 0);

DMA2_Stream3->PAR = (uint32_t)&SPI1->DR;

```

<figure 3.3.3>

이렇게 SPI와 DMA의 초기 설정을 마무리하면, 데이터 전송을 위한 사전 준비는 마무리

된다. 데이터를 전송을 위한 순서는 다음과 같다.

1. Interrupt flags disable (LIFCR)
2. DMA Stream disable
3. 송신 데이터를 배열에 저장
4. 전송 메모리 주소 할당 (M0AR)
5. 전송 데이터 크기 할당 (NDTR)
6. DMA Stream enable
7. 전송 완료 대기 (LISR_TCIF)

먼저 새로운 전송을 위해 이전 데이터 블록 DMA 전송에서 DMA_LISR의 스트림 전용 비트 설정을 해제해야 한다. 그리고 스트림을 비활성화한다. 만약 EN 비트가 1로 읽히는 경우, M0AR와 NDTR 레지스터에 대한 쓰기는 금지되기 때문에, EN 비트를 0으로 설정하여 DMA stream을 비활성화 시킬 필요가 있다. EN 비트가 0으로 읽히면, SPI가 제공한 데이터들을 따로 변수에 저장하고, 변수의 주소와 크기를 각각 M0AR와 NDTR 레지스터에 설정한다. 마지막으로 EN 비트를 SET하여 해당 Stream이 활성화되면 DMA 액세스가 요청된다. DMA는 그 후에 SPI_DR 레지스터에 쓰기 작업을 수행하고, 전송이 완료되면 LISR Register의 TCIF bit를 SET시킨다. 따라서 TCIF가 SET이 될 때까지 값을 계속 확인함으로써 쓰기과정의 이상이 없게끔 한다. 데이터 전송을 위해 사용된 레지스터에 대한 설명은 [table 3.3.4](#)에 나와있고, 작성된 코드는 [figure 3.3.4](#)에 나와있다.

	DMA stream x memory 0 address register (M0AR)	내용	
Bit 31:0	M0A[31:0] (Memory 0 address)	DMA 채널과 연결된 메모리의 주소를 설정하는 레지스터이다.	

	DMA stream x number of data register (NDTR)	내용	
Bit 15:0	NDT[15:0] (Number of data items to transfer)	DMA 채널에서 전송할 데이터의 개수를 설정한다.	

	DMA low interrupt flag clear register (DMA_LIFCR)	내용	
Bit 27	CTICIF3(Stream 3 clear transfer complete)	Stream3의 transfer	

	interrupt flag)	complete interrupt 를 clear해준다.
--	-----------------	--------------------------------

<table 3.3.4>

```
void DMA_Send(){
    DMA2->LIFCR |= DMA_LIFCR_CTCIF3 | DMA_LIFCR_CHTIF3 | DMA_LIFCR_CTEIF3 | DMA_LIFCR_CDMEIF3 | DMA_LIFCR_CFEIF3; // clear flags

    // disable EN bit
    DMA2_Stream3->CR &= ~DMA_SxCR_EN;
    while((DMA2_Stream3->CR)&DMA_SxCR_EN){};

    // WS2812 LED로 데이터 전송을 위한 한 줄 세우기
    int cnt = 0;
    for (int i = 0; i < numPixels * 3; i++) {
        for (int j = 7; j >= 0; j--) {
            TX_buf[cnt++] = (pixels[i] >> j) & 1 ? 0b11111000 : 0b11000000;
        }
    }

    DMA2_Stream3->M0AR = (uint32_t)TX_buf;
    DMA2_Stream3->NDTR=(192);

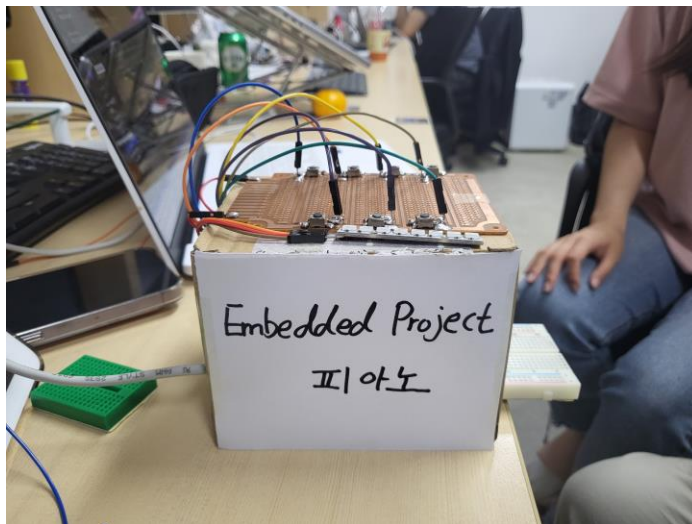
    // set EN bit
    DMA2_Stream3->CR |= DMA_SxCR_EN;
    while(!(DMA2->LISR & DMA_LISR_TCIF3)){
    }
}
```

<figure 3.3.4>

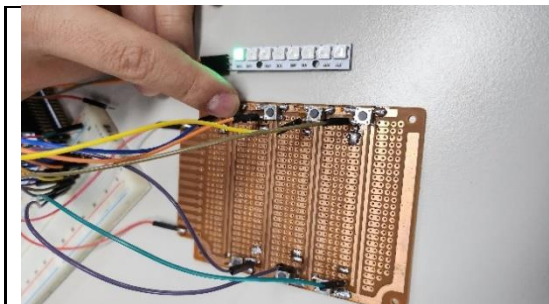
4. 결과 및 데모

회로 상의 특정 스위치를 눌렀을 때 DMA 및 SPI 통신을 이용하여 ~보드에서 WS2812B LED로 색상 제어 데이터를 전송하고, 피에조 부저로 PWM신호를 보내서 원하는 음계를 출력하는 데모용 최종 결과물은 figure 4.1 과 같다.

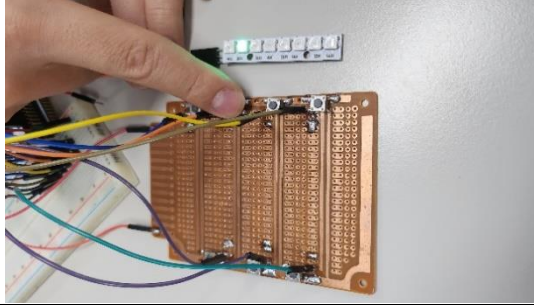
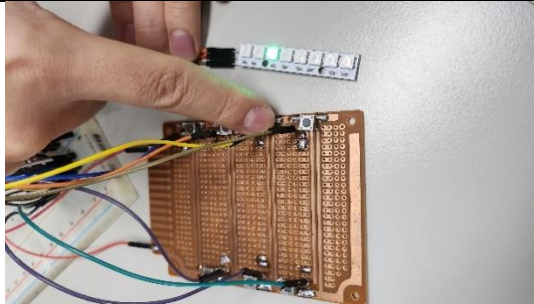
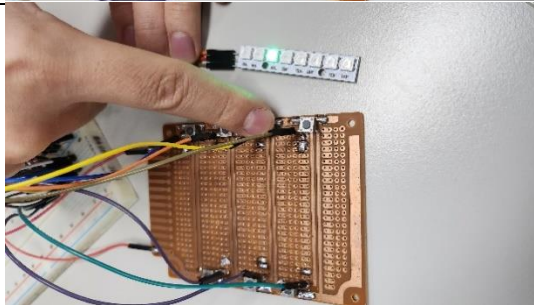
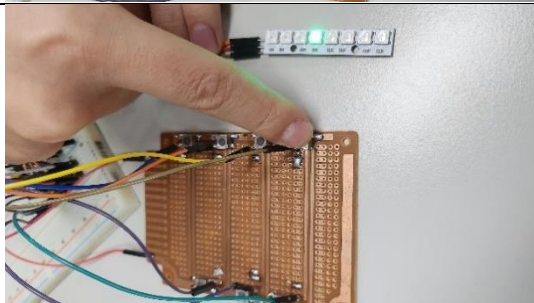
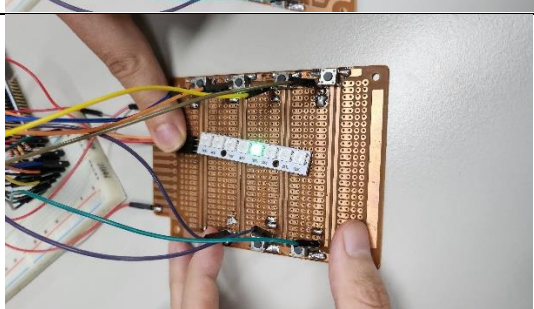
0번 스위치를 누르게 되면 ~보드의 D2핀으로 전압 값이 '1'로 읽히게 되고 이에 따라 음계 '도'가 피에조 부저에서 출력되고 DMA 및 SPI 통신으로 인해 WS2812 LED에서 첫번째에 해당하는 LED에 색상 값이 나타나게 된다. 동일한 스위치를 누르는 시간에 따라 색상의 밝기 값이 증가하게 되고, 임의의 임계 값에 도달하게 되면 $G \rightarrow R \rightarrow B$ 순서대로 색상이 변하는 식으로 동작한다. 이와 같은 방식으로 0~7번 스위치 모두 동작하는 것을 알 수 있다.

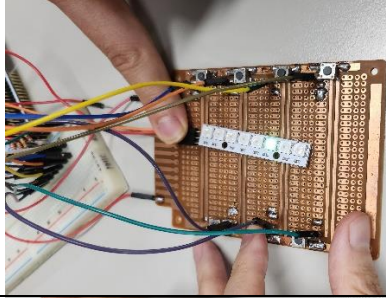
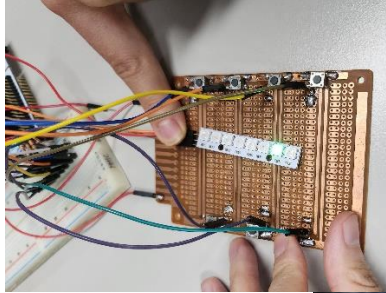
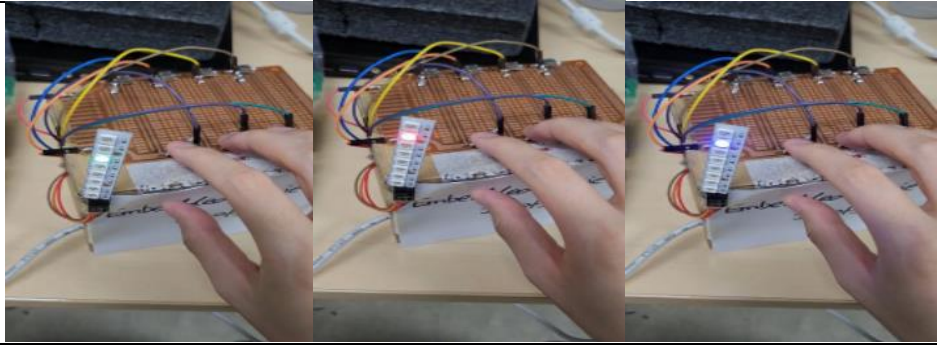


<figure 4.1_최종 데모용 결과물>



0번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인

	<p>1번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	<p>2번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	<p>3번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	<p>4번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	<p>5번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>

	<p>6번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	<p>7번 스위치 ON시 첫 번째 LED에 색상이 뜨는 것을 확인</p>
	
<p>5번 스위치의 상태를 ON으로 유지한 경우, WS2812B LED의 색상이 G → R → B 순서대로 출력됨을 확인</p>	

<figure 4.2>

5. 결론

(1) 한계점

WS2812에 대해서 한계점이 존재했다. WS2812의 LED를 ON/OFF 해주기 위해서는 각 LED에 해당하는 RGB값을 보내주어야 한다. 우리가 생각하기엔 8개의 LED의 정보를 보내기 위해선 192비트가 필요했지만, WS2812는 1과 0을 인식하기 위해서 8비트의 값이 필요하여 결과적으로 192바이트가 필요하였다. 또한 비트 사이의 송신 시간 또한 1과 0을 받아들이는 데에 달랐다. 즉 정확한 타이밍을 계산하기 위해선 비트별로 보내는 타이밍을 딜레이 등과 같은 타이머를 이용하여 정확하게 컨트롤 해야 한다. 하지만 SPI로 보내는 조건에 따라 데이터 송신마다 딜레이를 이용할 수는 없었다. 이를 해결하기 위해선 SPI의 Baud Rate를 우리가 원하는 타이밍 조건을 만족하도록 설정하는 것이었다. 이는 오차를 감안한 설정이기 때문에 부정확 할 수 있다는 한계점을 지니고 있다.

다음으로는 Push Button에 대해 한계점이 존재했다. 피아노를 구현하기 위해 버튼을 사용하였다. 이 버튼의 출력은 오직 0과 1이며 눌리거나 안눌리거나다. 하지만 모두가 알듯이 피아노는 세게 누르면 소리가 크게 난다. 소리의 크기는 푸쉬버튼을 이용해서는 구현할 수 없다는 기술적 한계가 있었다. 이를 해결하기 위해선 무게 센서로 눌린 무게를 감지하여 세기를 측정하거나 얼마나 빨리 눌렀는지를 측정하기 위한 센서를 이용하는 방법이 있겠지만, 기술적 난이도로 인하여 푸쉬버튼을 이용하게 되었다.

마지막으로 Piezo buzzer를 사용하는 부분에서 한계점을 마주하였다. 피에조에서는 해당하는 건반이 눌리면 이에 해당하는 음이 나온다. 하지만 두 개 이상의 입력에 대해선 뒤늦게 눌린 입력만 소리가 나온다. 즉 동시에 두 개의 음에 대한 출력은 할 수 없었다. 피에조 특성상 하나의 주파수에만 대응할 수 있기에 당연한 결과였다.

(2) 총평

본 프로젝트는 단계별로 밟아 나가는 방식으로 진행하였다. 먼저는 푸시 스위치 버튼을 시작으로 버튼 입력에 따라 피에조 부저 작동, SPI를 이용한 WS2812점등, 마지막으로 DMA와 SPI를 함께 사용하여 WS2812점등을 검증하며 진행하였다. 사실 처음부터 DMA를 이용한 통신에 도전하였다. 하지만 전체적인 흐름과 그 안의 세부적인 동작에 대한

이해가 없이 진행하다 보니 전진할 수 없었다. 결국 가장 빠른 길은 처음부터 시작하는 것임을 깨닫게 되어 위와 같은 순서로 검증 단계를 진행하였다. 이를 통해 각 단계별 중점 사항을 체크할 수 있었고 전체적인 흐름에 대한 시야가 생기게 되었다. 그 사항들은 다음과 같다.

푸시 스위치에 대한 공부와 이해를 경험할 수 있었다. 일반적으로는 스위치를 누르면 '1', 그렇지 않으면 '0'이라는 1차원적인 결과를 기대했다. 하지만 실제로 진행해보니 우리가 기대한 결과와 매우 다른 모습을 볼 수 있었다. 문제에 대해 고민과 시도 끝에 플로팅 현상이라는 개념에 도달할 수 있었다. 이에 대해 풀다운 저항을 회로에 연결하는 방법으로 문제를 해결할 수 있었다. 문제 상황을 해결하기 위해 고민하는 시간들을 통해 안정적인 결과를 낼 수 있었다.

또한 레지스터를 bit단위로 직접 제어하는 것에 대해 공부할 수 있었다. 기존 수업의 과제에서는 MBED-OS에서 제공하는 라이브러리를 이용하여 SPI객체를 생성하고 이를 이용하여 SPI통신을 진행하였기 때문에 레지스터를 직접 제어하는 것이 쉽지만은 않았지만 함수에 의존하지 않고 datasheet를 직접 이해하고 조절해보면서 정상적으로 작동하는 결과를 낼 수 있었던 것이 큰 성취감과 배움을 느끼게 하였다.

6.참조

<https://m.blog.naver.com/emperonics/221725399383>

<https://news.samsungdisplay.com/27245>

<https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=arduinosealer&logNo=20864624712>

<https://stemwith.github.io/2018/09/07/Arduino-Piezo-Buzzer/>

<https://kocoafab.cc/tutorial/view/539https://kocoafab.cc/tutorial/view/539>

<https://gammabeta.tistory.com/3320https://gammabeta.tistory.com/3320>

<STM32F411xC/E advanced Arm®-based 32-bit MCUs - Reference manual>

<https://cdn-shop.adafruit.com/datasheets/WS2812.pdf>

<https://www.hackster.io/RVLAD/neopixel-ws2812b-spi-driver-with-ada-on-stm32f4-discovery-d330ea>

<https://k96-ozon.tistory.com/59>