소프트웨어 마에스트로 11기 FAST팀   
임베디드 기기 펌웨어 매뉴얼

**작성자: 손승우**

**최종 작성일: 2021.02.15**

# 목차

[사용 프로그램 설치 과정 및 예제 실행 과정 1](#_Toc64278891)

[1. Keil uVision5 1](#_Toc64278892)

[2. nRF5 SDK v13.0.0 4](#_Toc64278893)

[3. nRF Connect 6](#_Toc64278894)

[4. nRF5 SDK v13.0.0 blinky 예제 테스트 8](#_Toc64278895)

[프로그램 실행 방법 및 코드 상세 설명 14](#_Toc64278896)

[1. 프로그램 실행 방법 14](#_Toc64278897)

[2. 프로그램 기본 개요 17](#_Toc64278898)

[3. 장갑 펌웨어 ADC 코드 설명 18](#_Toc64278899)

[4. 장갑 펌웨어 MUX 코드 설명 22](#_Toc64278900)

[5. 장갑 펌웨어 TWI(I2C) / IMU 코드 설명 25](#_Toc64278901)

[6. 장갑 펌웨어 BLE 코드 설명 39](#_Toc64278902)

[7. 장갑 펌웨어 전체 구성 43](#_Toc64278903)

# 사용 프로그램 설치 과정 및 예제 실행 과정

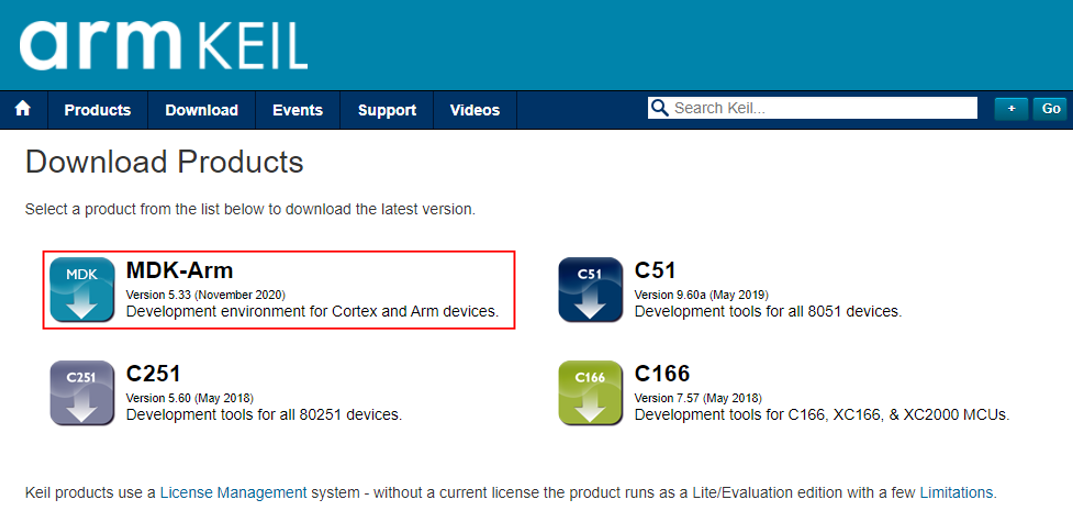
## 1. Keil uVision5

* 펌웨어의 컴파일, 디버깅 등을 할 수 있는 IDE
* 다운로드 링크: <https://www.keil.com/>

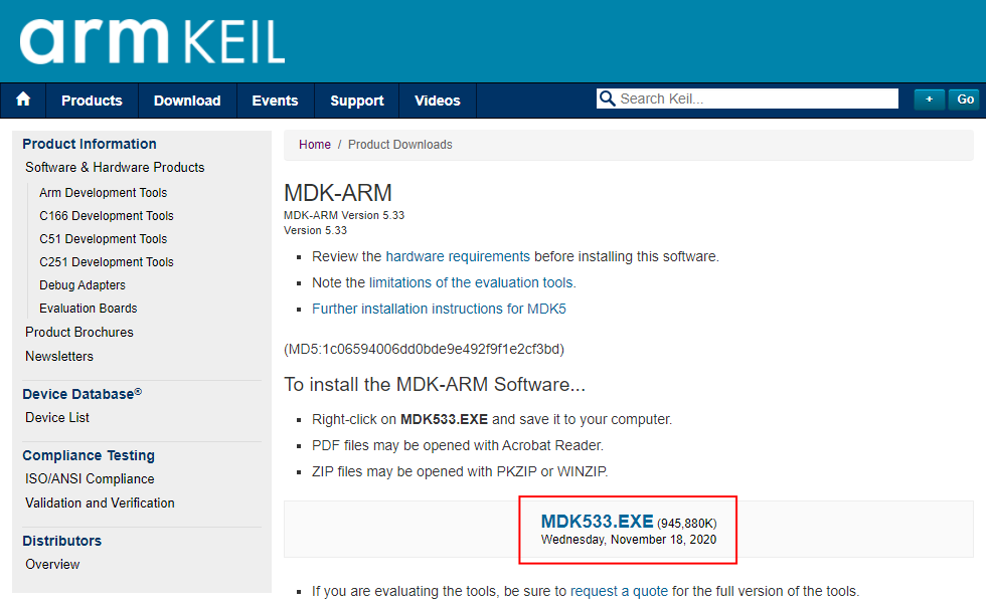
1. Download 클릭



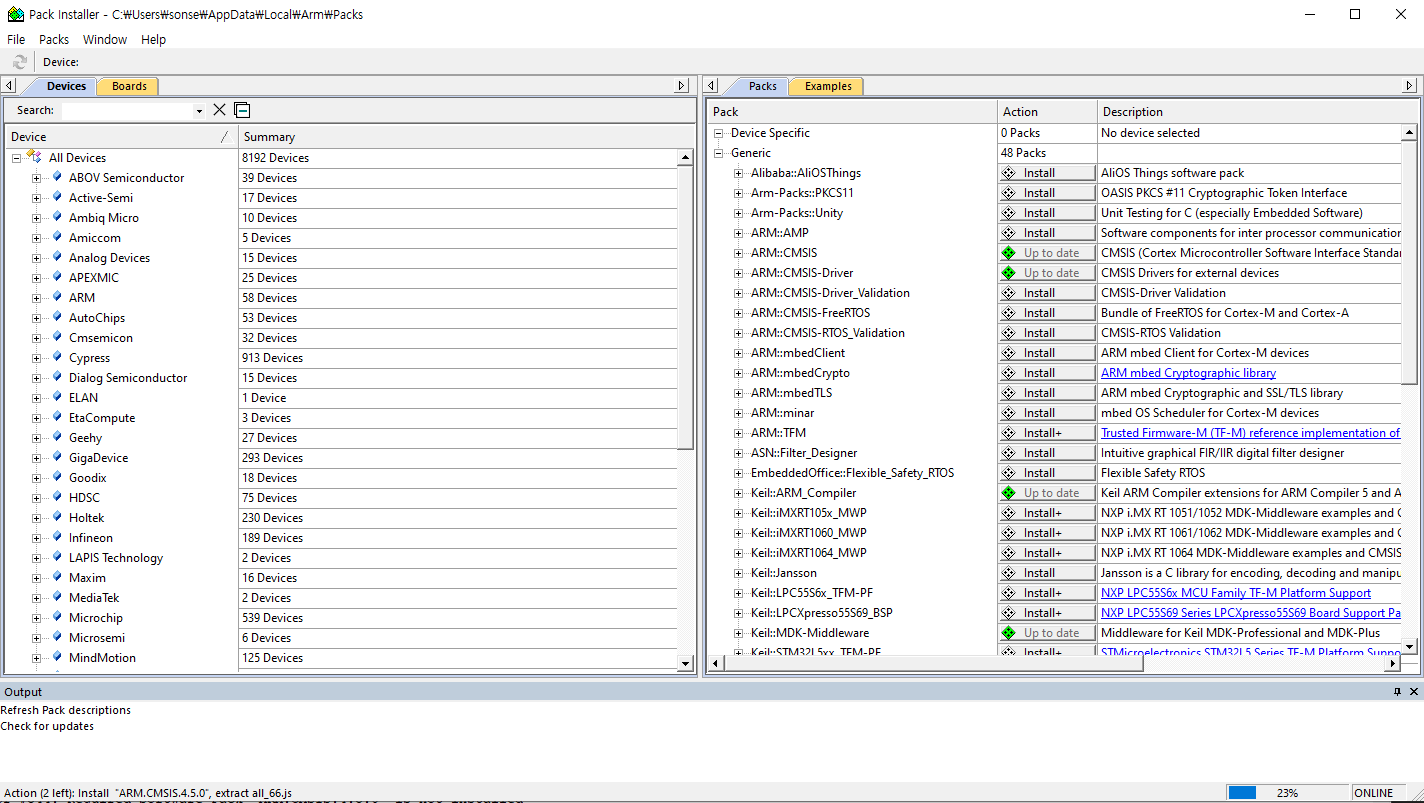
1. MDK-Arm 선택



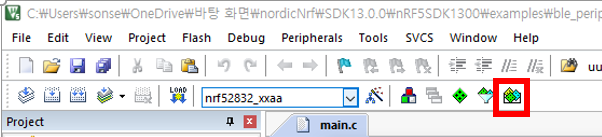
1. MDK-Arm 선택 시 나오는 정보 입력창에 본인 정보를 입력하고 설치 파일 다운로드



1. 설치가 완료되면 다음과 같은 창이 뜨는데 다운로드가 완료될 때까지 대기



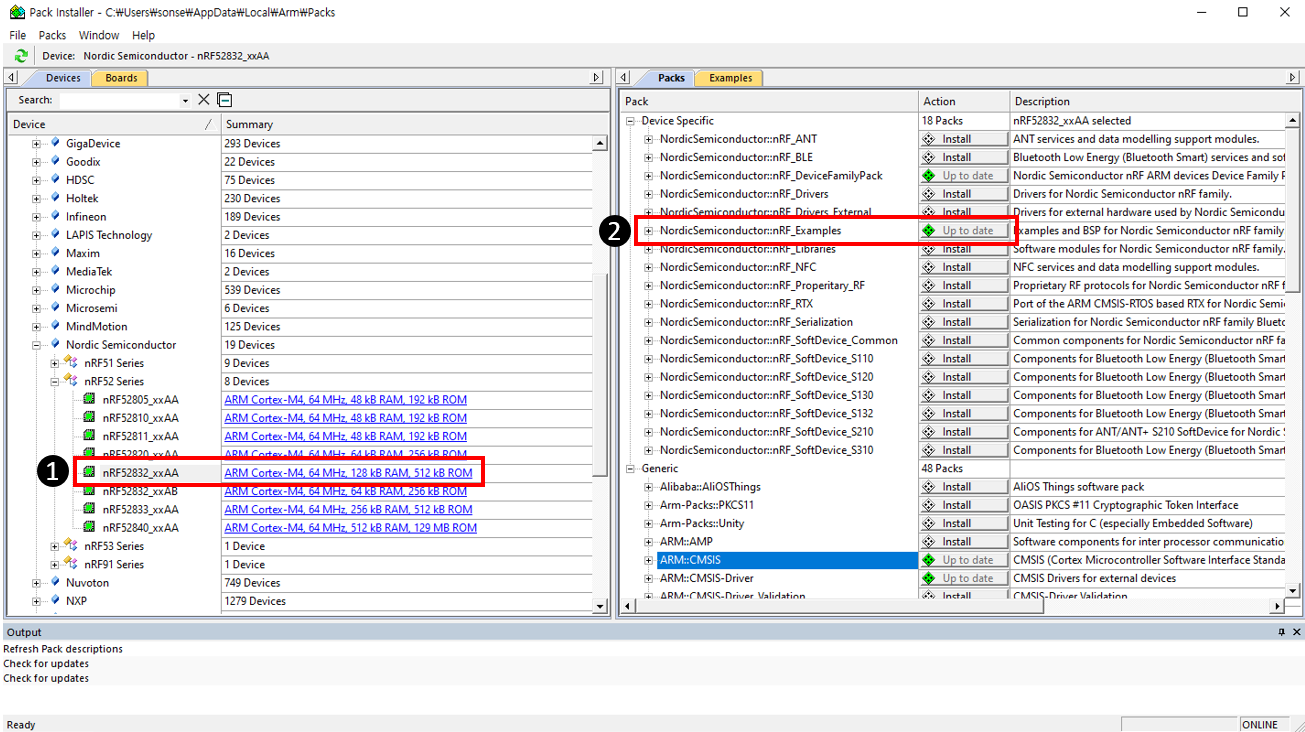
1. 해당 창은 Pack Installer로 펌웨어 제작에 필요한 각종 driver, header 파일 및 예제 파일 등을 다운받을 수 있고 keil project 화면에서 pack installer 버튼을 눌러 열 수 있음



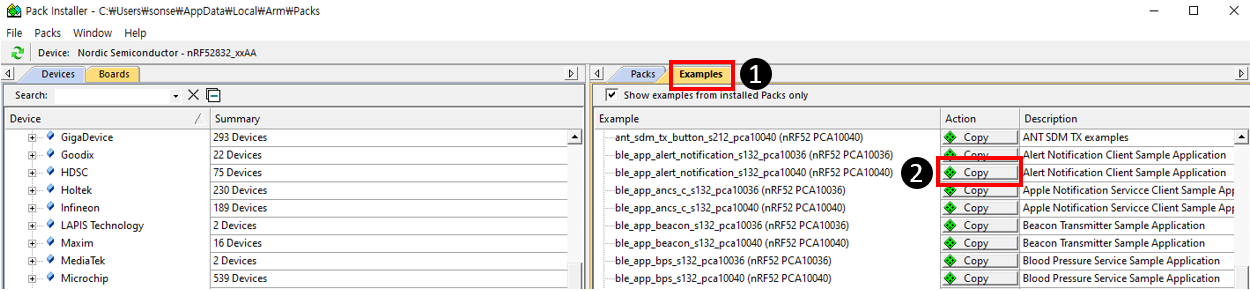
1. 예제 다운로드

Pack Installer로 예제를 다운 받아서 사용할 수 있지만 보통 뒤에서 설명하는 SDK를 이용하여 개발

1. Pack Installer 왼쪽 화면 Devices에서 사용하려는 칩 선택 후 오른쪽 화면 Packs에서 Examples를 Install



1. Packs 버튼 옆에 Examples 버튼을 눌러 원하는 예제 다운



## 2. nRF5 SDK v13.0.0

* Nordic사 nRF5xx 계열 펌웨어 개발 시 요구되는 SDK

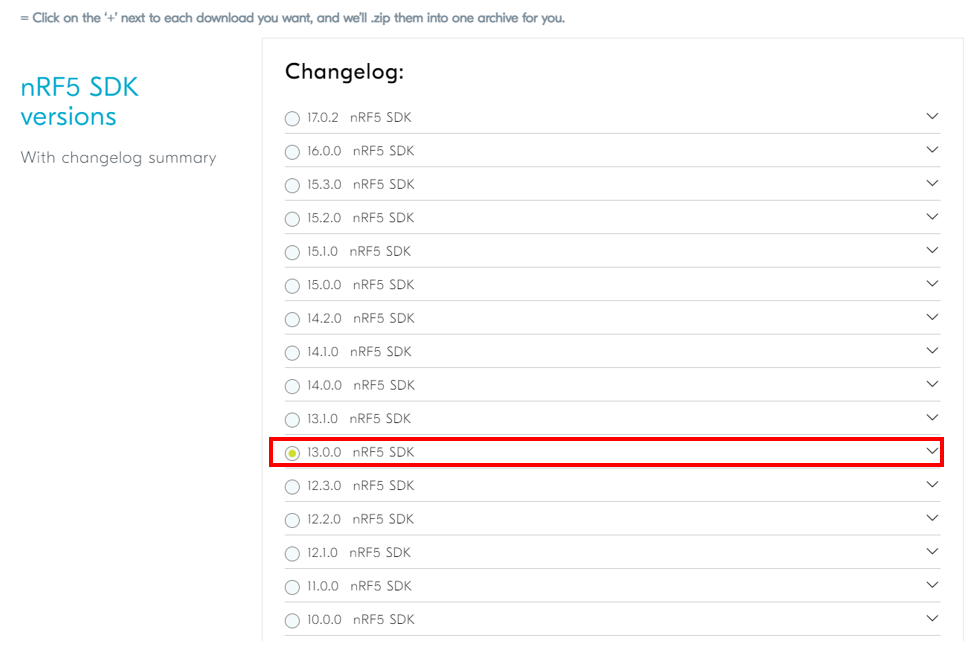
(공식 문서: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)

* 다운로드 링크: <https://www.nordicsemi.com/Software-and-tools/Software/nRF5-SDK/Download#infotabs>

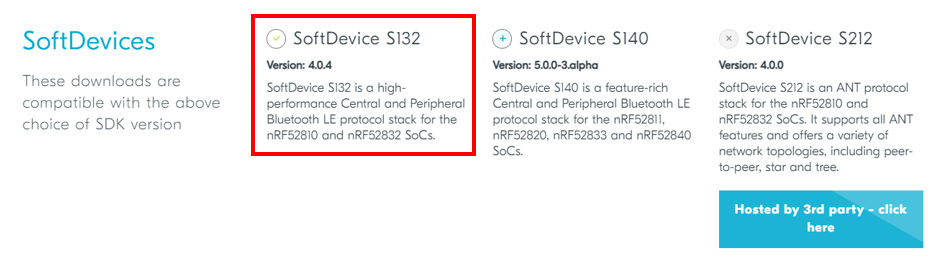
1. Download 클릭



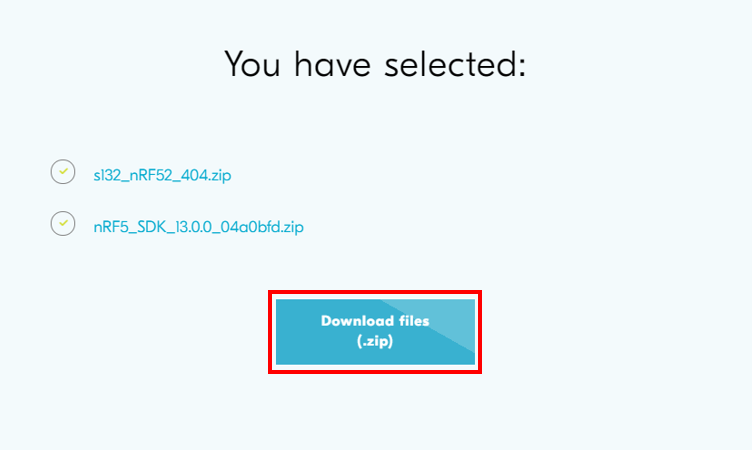
1. 스크롤을 내려 13.0.0 nRF5 SDK를 선택



1. 스크롤을 내려 SoftDevice S132 선택



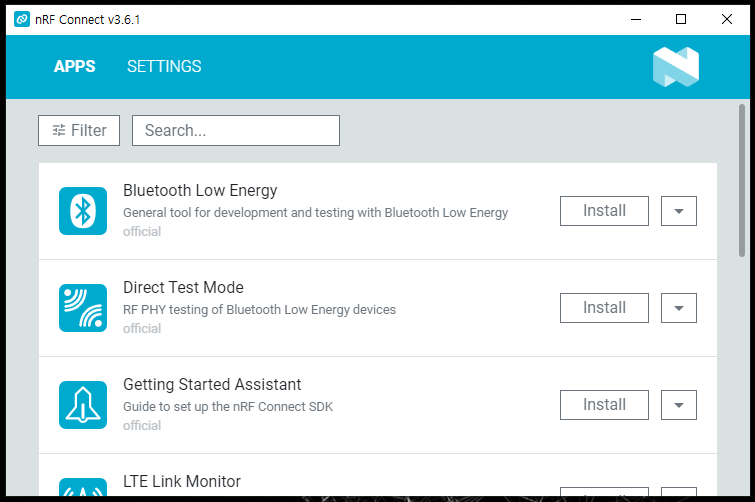
1. 스크롤을 내려 Download files를 클릭하여 압축 파일을 다운로드하고 압축 해제



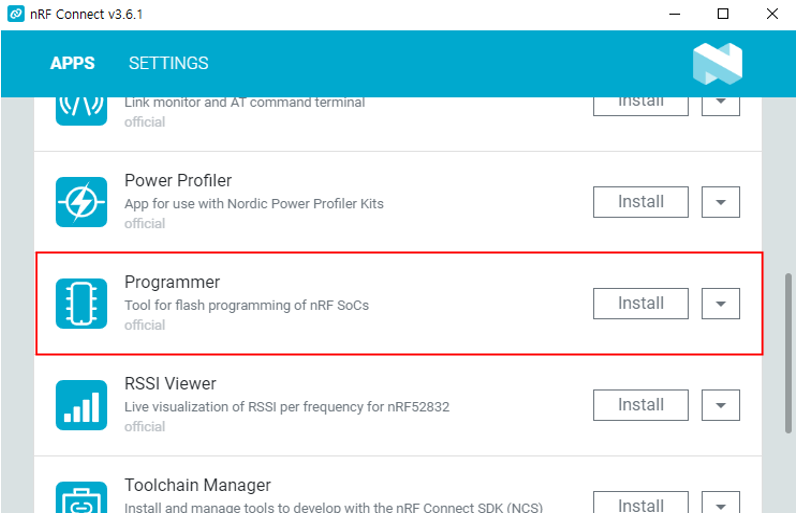
## 3. nRF Connect

* Nordic사 보드를 제어하거나 펌웨어를 write하는데 사용되는 개발 툴
* 다운로드 링크: <https://www.nordicsemi.com/Software-and-tools/Development-Tools/nRF-Connect-for-desktop/Download#infotabs>

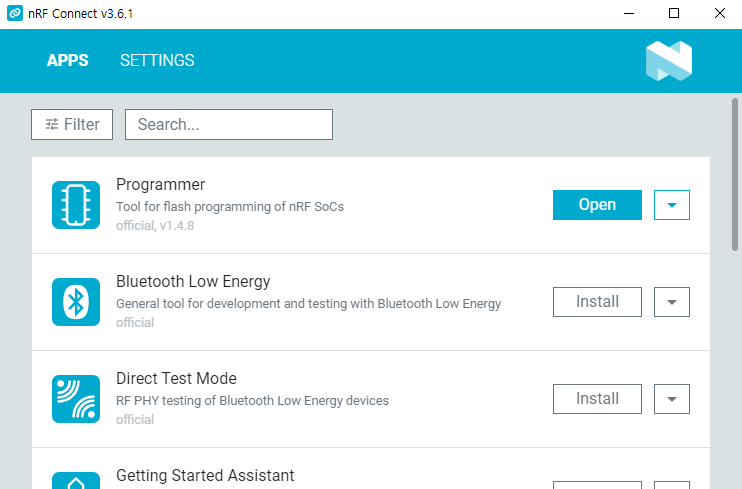
1. 설치 후 첫 실행 화면

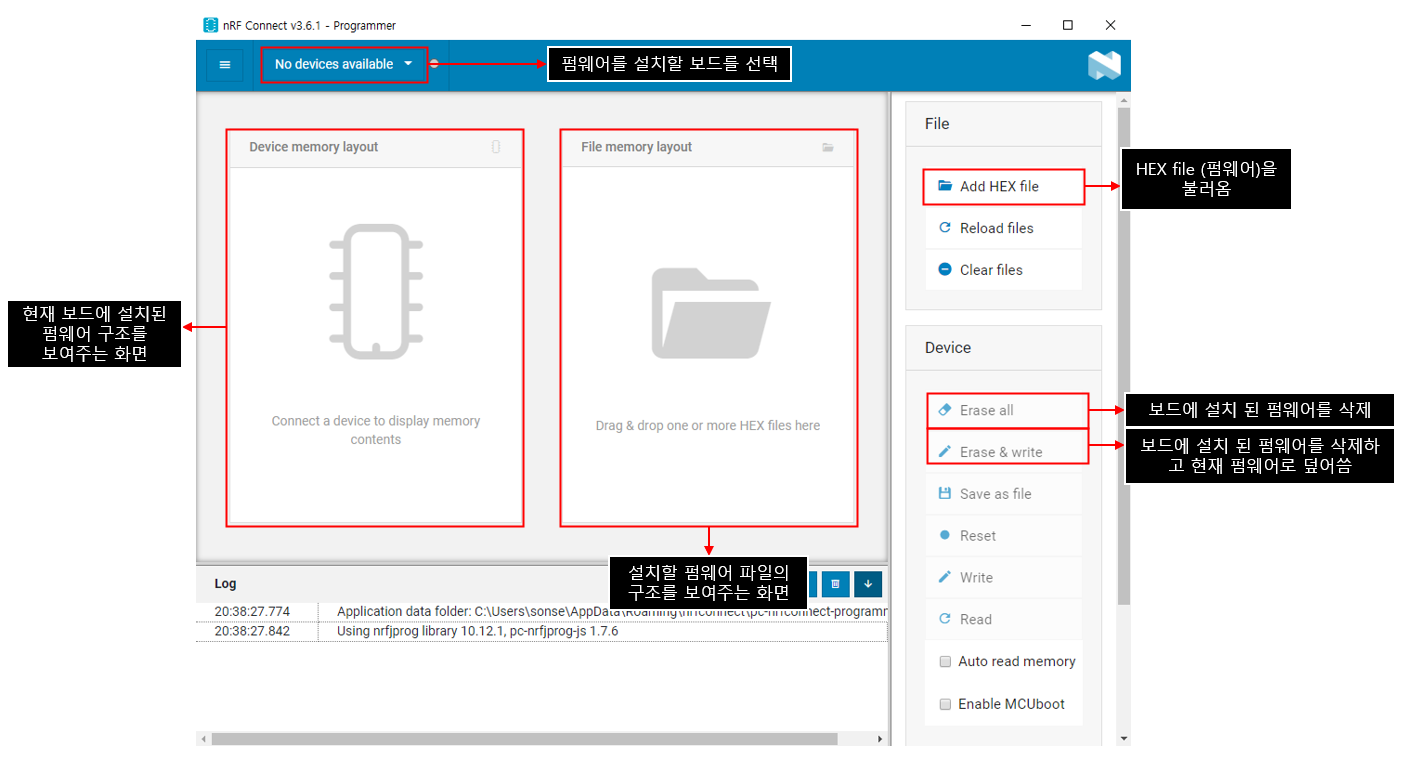


1. 스크롤을 내려 Programmer를 Install



1. Programmer 설치 후 화면



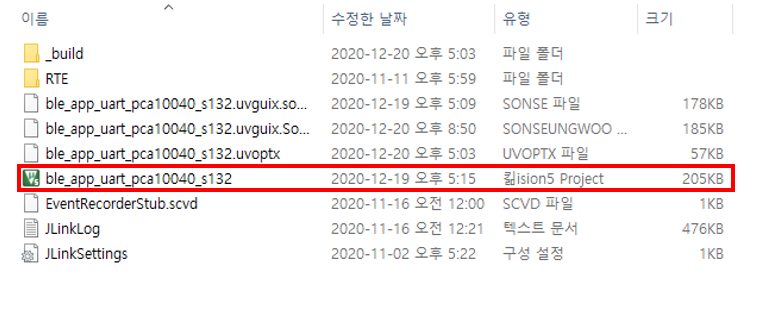
1. Programmer 실행 화면 및 구성

## 4. nRF5 SDK v13.0.0 blinky 예제 테스트

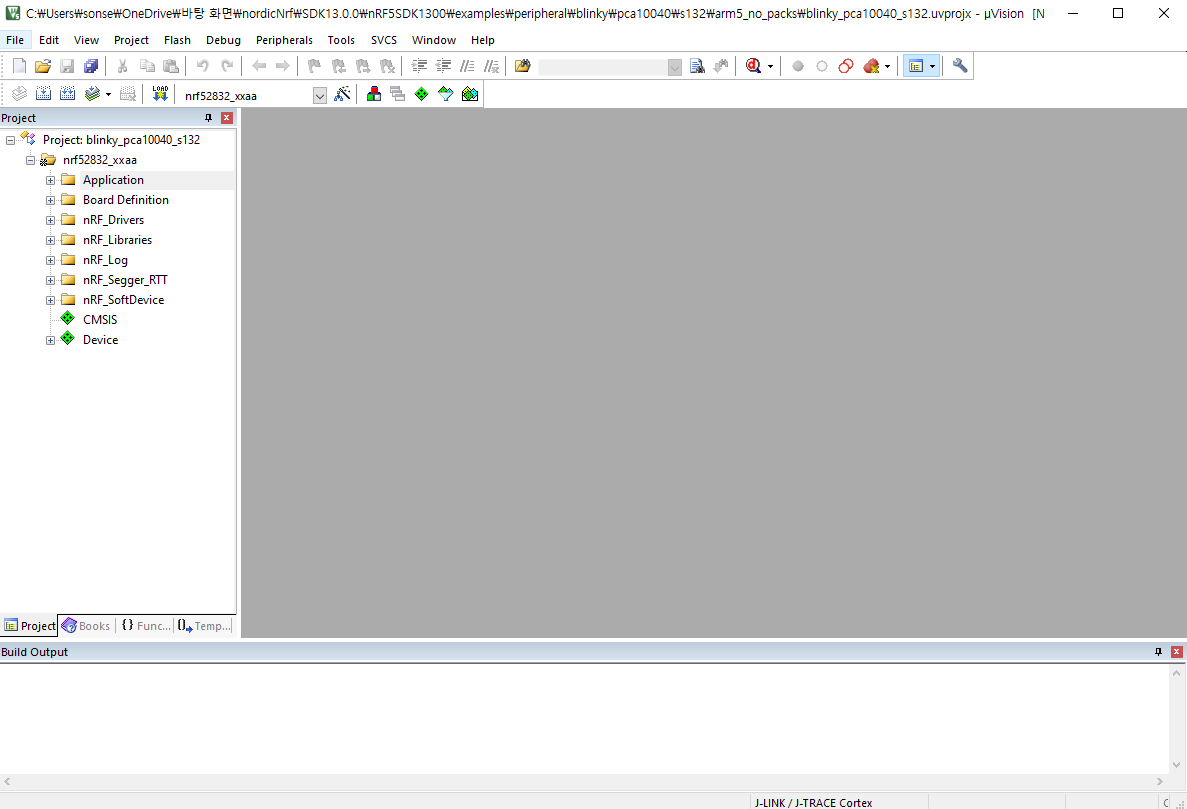
* 보드에 장착된 led가 순서대로 켜졌다가 꺼지는 것을 반복하는 예제
* SDK의 예제 파일을 keil로 펌웨어 빌드하고 nRF Connect를 사용하여 보드에 write하기
  1. 다운로드한 SDK 폴더에서 keil 프로젝트 파일 열기

(파일 위치: SDK13.0.0\nRF5SDK1300\examples\peripheral\blinky\pca10040\s132

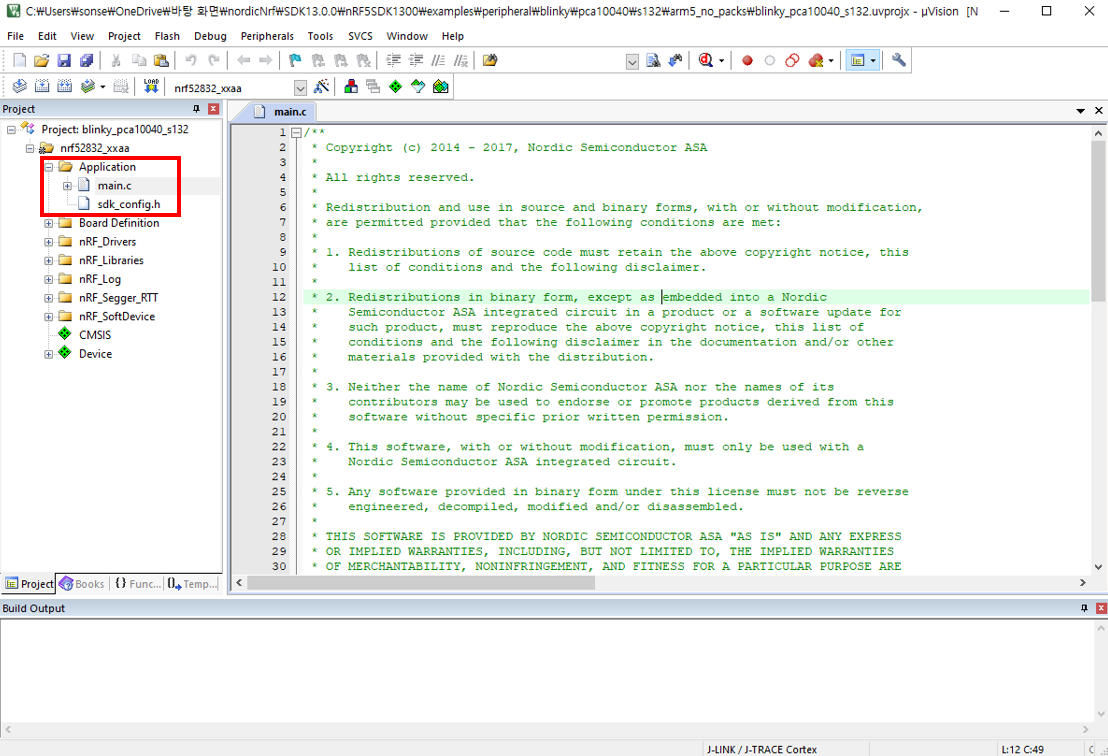
\arm5\_no\_packs)



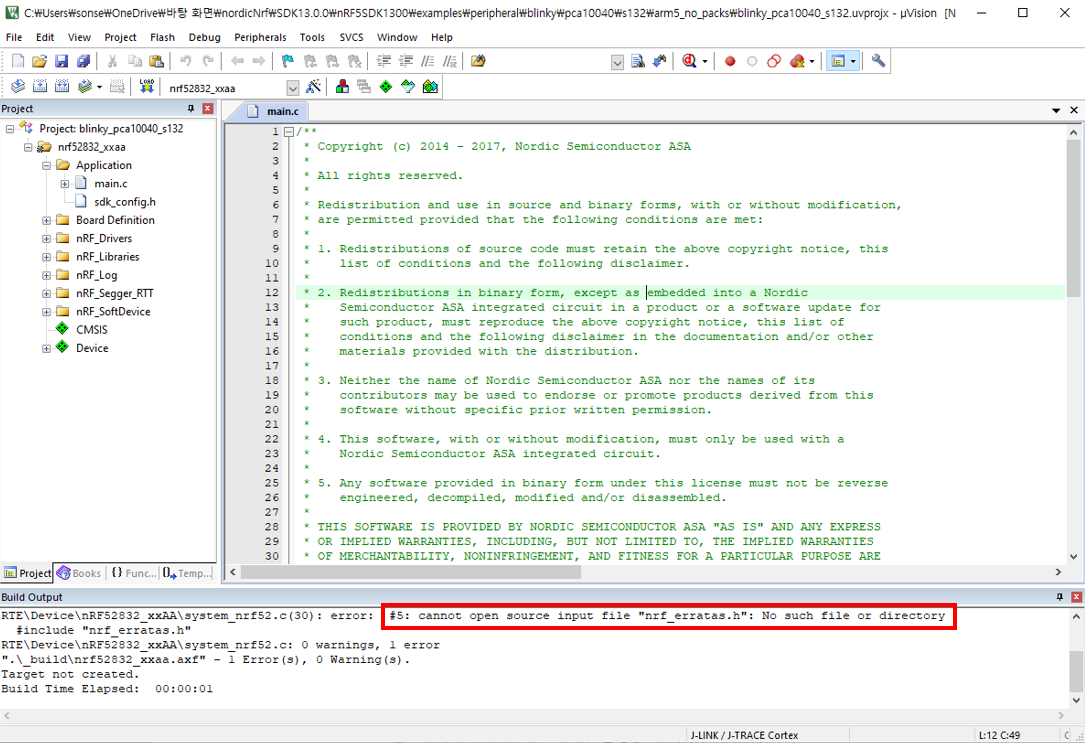
* 1. 프로젝트 파일을 열면 다운로드하라는 알림 창이 뜨는데 모두 다운로드, 다운로드가 완료되면 다음과 같은 프로젝트 화면이 나옴



* 1. Application 폴더 안의 main.c 파일을 선택하여 main 파일을 불러옴



* 1. F7 버튼을 눌러 빌드

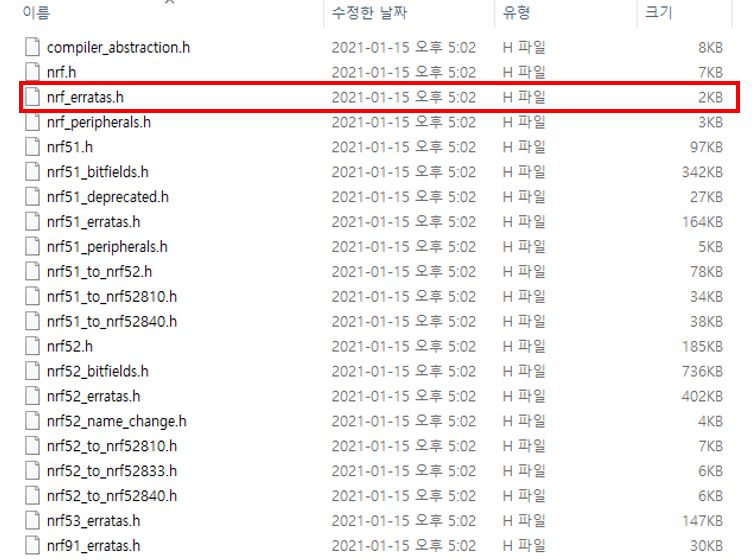


\* 아마 cannot open source input file “nrf\_erratas.h”: No such file or directory 라는 에러가 뜰 텐데 파일 경로 설정을 해줘야 함

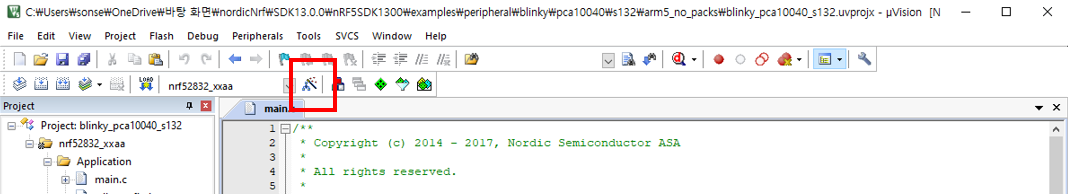
* 1. nrf\_erratas.h 위치

보통 “C:\Users\사용자명\AppData\Local\Arm\Packs\NordicSemiconductor

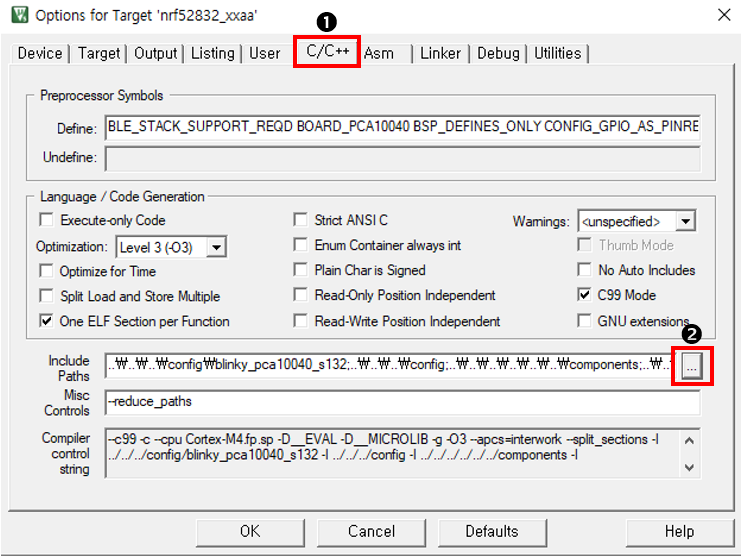
\nRF\_DeviceFamilyPack\8.38.0\Device\Include”에 위치



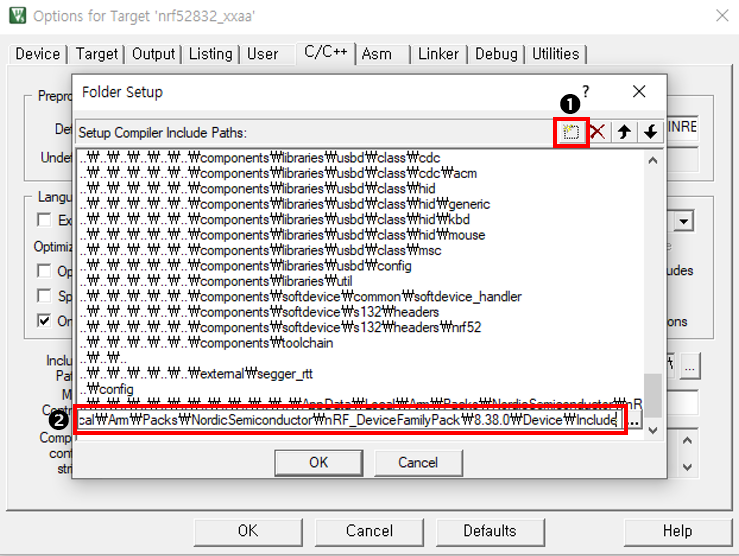
* 1. include path 설정
     1. option 열기



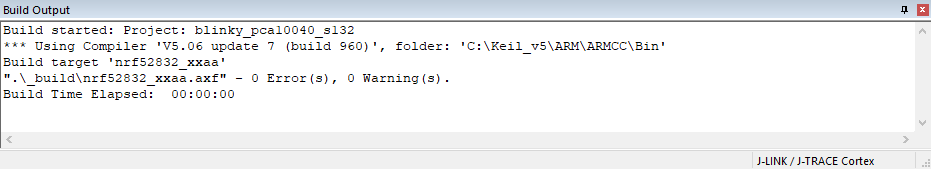
* + 1. Folder setup 메뉴 열기



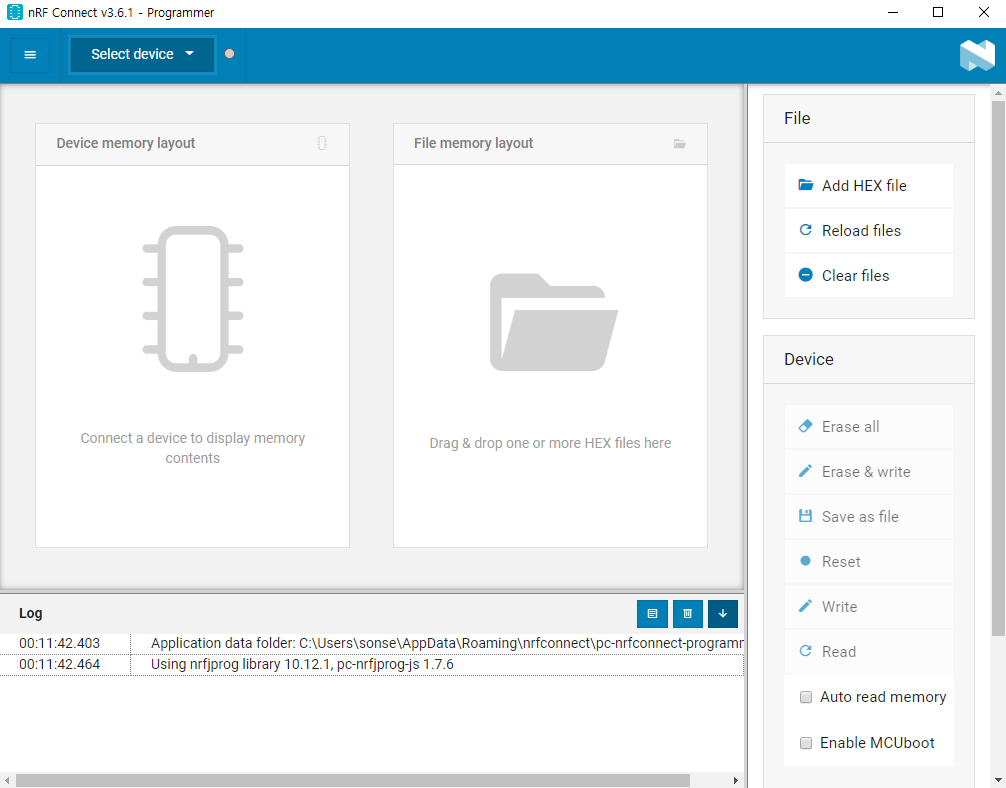
* + 1. nrf\_erratas.h 경로 추가



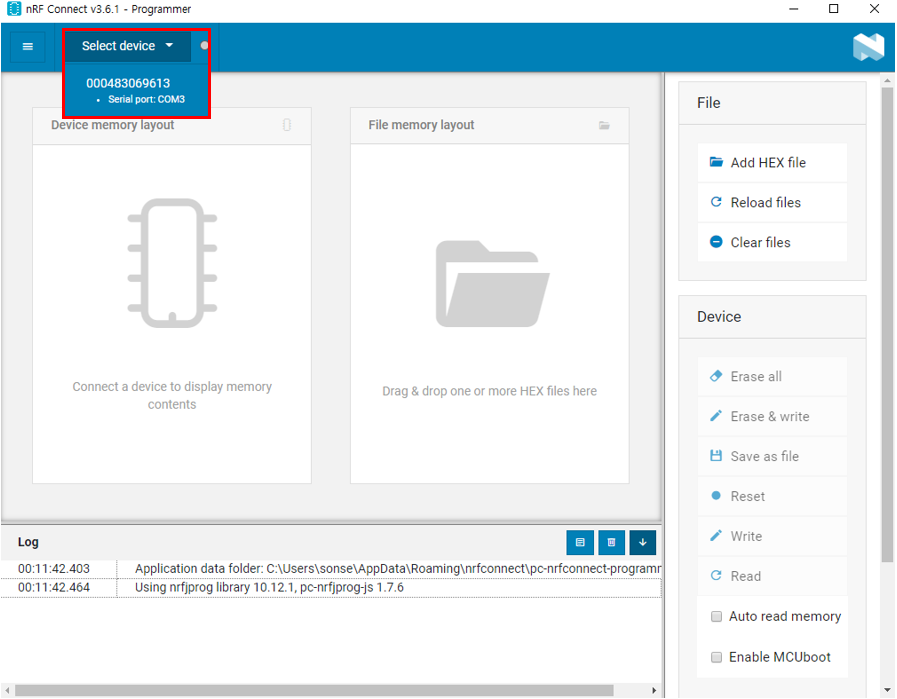
* 1. 경로 추가가 완료되었으면 다시 빌드



* 1. 보드를 컴퓨터에 연결하고 nrf Connect의 programmer를 실행

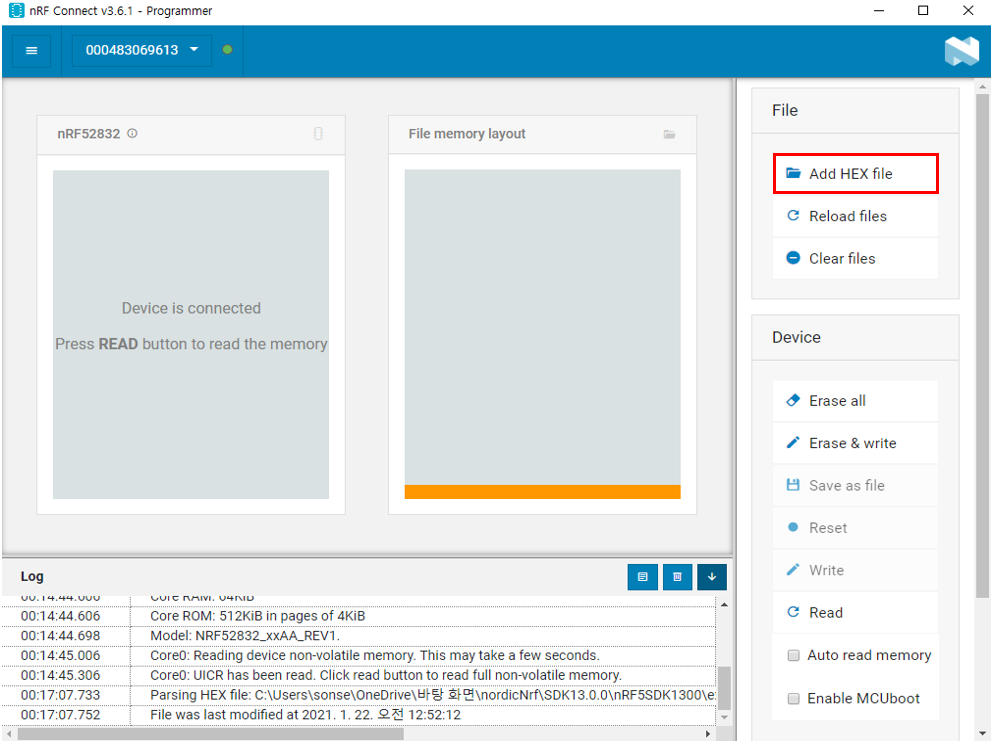


* 1. Select device 버튼을 눌러 보드를 선택

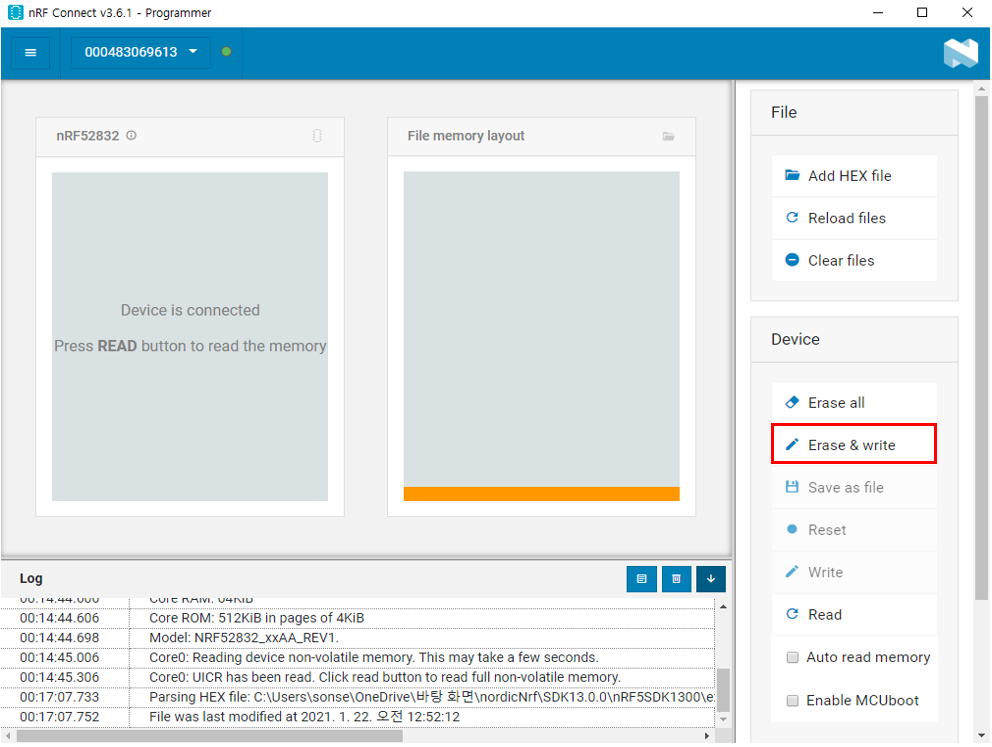


* 1. Add Hex file 버튼을 눌러서 keil로 빌드한 파일을 선택

파일 위치는 keil 프로젝트가 위치한 폴더 안의 \_build 폴더에 있음



* 1. Erase & write 버튼을 눌러 보드에 펌웨어 write



* 1. 보드의 LED가 차례대로 켜졌다가 꺼지는 것을 반복하는 걸 확인하여 펌웨어가 제대로 write되었는지 확인



# 펌웨어 실행 방법 및 코드 상세 설명

## 1. 펌웨어 기본 개요

**1) 스마트 글러브 펌웨어**

* flex 센서, IMU로 손가락/어깨 굽힘, 손목/팔꿈치 회전을 측정하고 측정 값들을 BLE를 통해 전송하는 스마트 글러브를 동작 시키기 위한 펌웨어 (<https://github.com/SeungwooSon225/SM11_FAST.git>)
* 크게 4가지 기능으로 나눌 수 있음

1. ADC: flex 센서 측정을 위해 사용

참고 예제: nRF5 SDK v13.0.0 peripheral/saadc  
(예제 개요: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)  
(공식 API: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)

1. MUX: flex 센서가 총 10개 사용되는데 이는 보드의 ADC 채널 수보다 많은 수이므로 MUX를 사용, 10개의 flex 센서를 하나의 MUX에 5개씩 연결하여 총 2개의 MUX를 사용

참고 예제: SparkFun Multiplexer Breakout - 8 Channel Arduion Example  
(<https://github.com/sparkfun/74HC4051_8-Channel_Mux_Breakout>)

1. I2C/IMU: IMU 사용을 위해 I2C 통신을 사용

참고 예제1: nRF5 SDK v13.0.0 peripheral/twi\_sensor  
(예제 개요: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)  
(공식 API: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)

참고 예제2: SparkFun BNO080 Arduino Example  
(<https://github.com/sparkfun/SparkFun_BNO080_Arduino_Library>)

1. BLE: 측정한 값들을 전송하기 위해 BLE를 사용

참고 예제: nRF5 SDK v13.0.0 ble\_peripheral/ble\_app\_uart  
(예제 개요: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)  
(공식 API: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)

**2) BLE 수신용 보드 펌웨어**

* 장갑에서 BLE로 전송한 측정 값들을 받아서 PC에 UART를 통해 전송하기 위한 펌웨어
* SDK의 ble\_app\_uart\_c 코드에서 BLE\_GATT\_ATT\_MTU\_DEFAULT 값만 43으로 변경하여 사용, 변경 방법은 [**6. 스마트 글러브 펌웨어 BLE 코드 설명**을](#_6._스마트_글러브) 참고

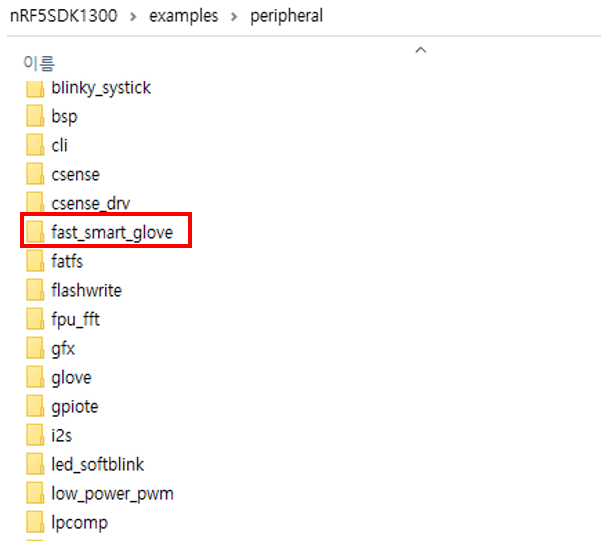
참고 예제: nRF5 SDK v13.0.0 ble\_central/ble\_app\_uart\_c  
(예제 개요: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)  
(공식 API: <https://infocenter.nordicsemi.com/index.jsp?topic=%2Fcom.nordic.infocenter.sdk5.v11.0.0%2Fgroup__nrf__saadc__hal.html>)

## 2. 펌웨어 업로드 방법

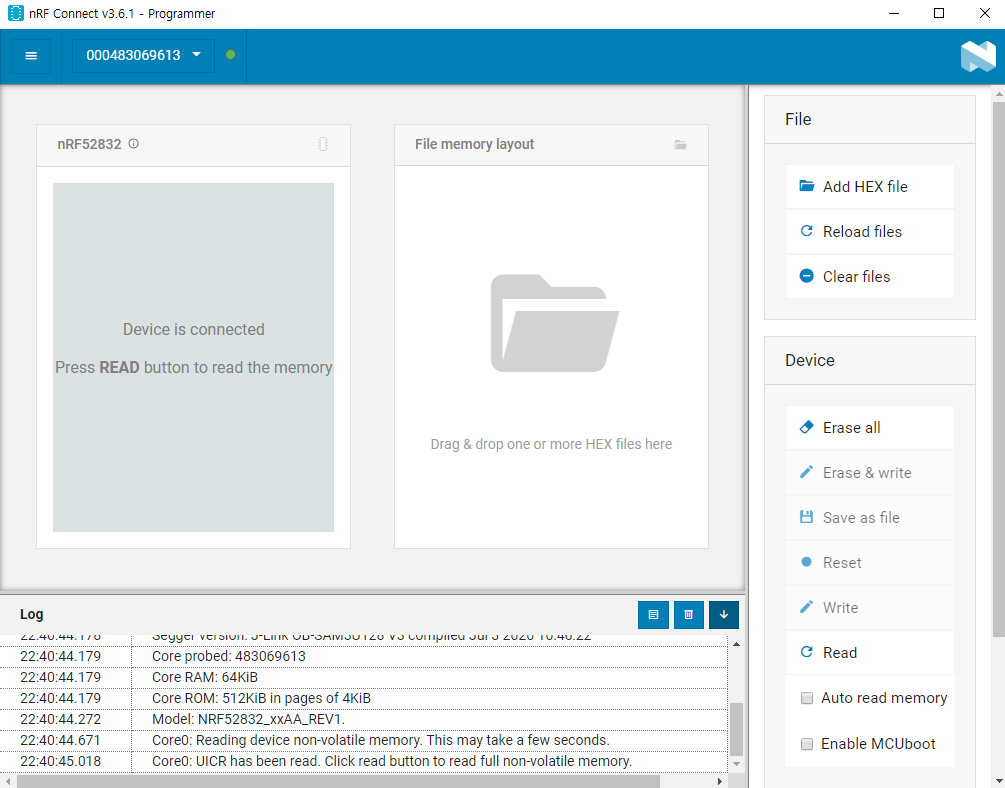
**1) 스마트 글러브 펌웨어 업로드**

1. 코드를 다운 받아서 SDK 폴더 안의 examples\peripheral 경로로 복사

다운 주소: <https://github.com/SeungwooSon225/SM11_FAST.git>

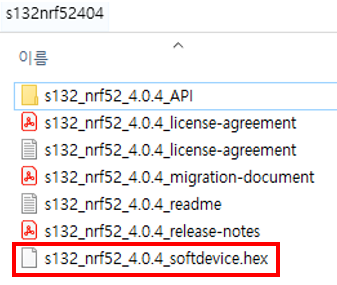


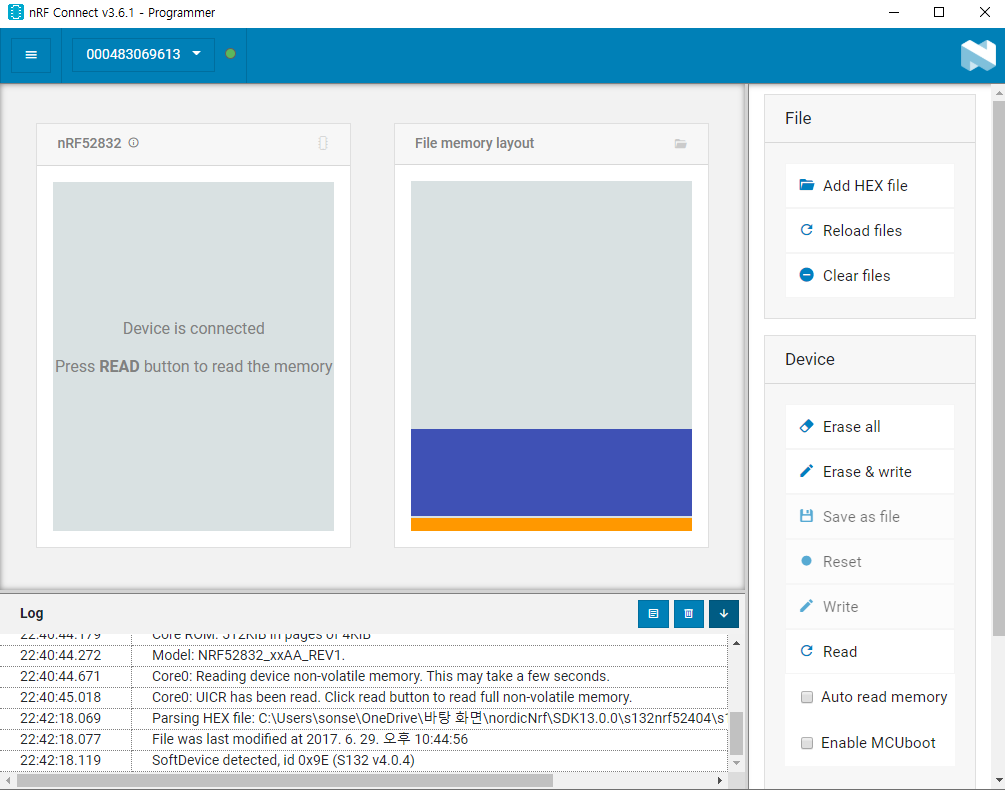
1. fast\_smart\_glove\pca10040\s132\arm5\_no\_packs 위치의 keil 프로젝트 파일을 실행 후 빌드, 이 때 예제를 실행할 때와 같이 nrf\_erratas.h 관련 에러가 뜬다면 같은 방법으로 해결
2. 보드를 컴퓨터에 연결하고 nrf Connect의 programmer를 실행 후 보드 선택



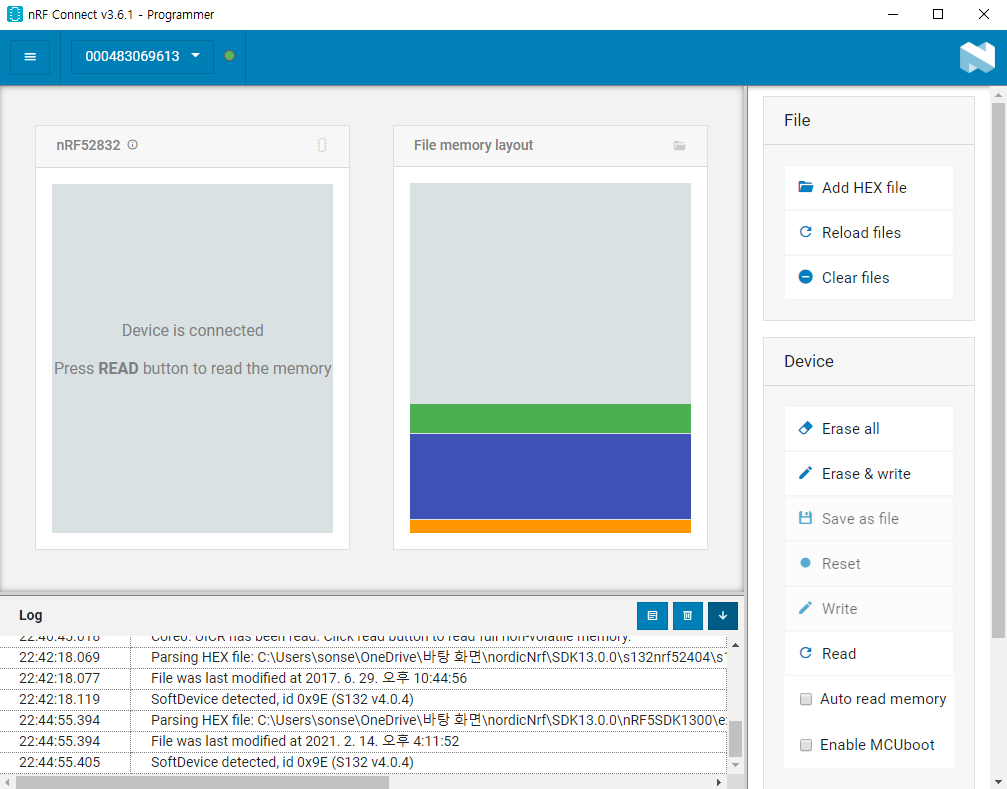
1. Add HEX file을 눌러 SoftDevice를 선택

SoftDevice는 SDK와 함께 다운 받은 폴더 안에 .hex 파일 형태로 존재





1. Add HEX file을 눌러 빌드한 hex파일을 선택



1. Erase & write 버튼을 눌러 보드에 펌웨어 write

**2) BLE 수신용 보드 펌웨어 업로드**

1. BLE\_GATT\_ATT\_MTU\_DEFAULT 값을 수정한 ble\_app\_uart\_c 예제를 빌드 하여 업로드
2. 상세 업로드 과정은 장갑 펌웨어 업로드 과정과 동일

## 3. 스마트 글러브 펌웨어 ADC 코드 설명

**1) 기본 개념**

**(1) Digital Output**

Analog input에 들어오는 analog 값에 대해 다음과 같이 Digital Output을 계산



* V(P): positive input의 voltage값
* V(N): negative input의 voltage값
* GAIN: 사용자가 설정한 GAIN값 (SAADC\_GAIN1\_2일 경우 GAIN = 1/2)
* RESOLUTION: 사용자사 설정한 RESOLUTION값
* m: config.mode=single ended -> 0, config.mode=differential -> 1

(우리는 ground를 negative input으로 사용하므로 single ended mode이고 m=0)

* REFERENCE

Reference는 Internal Reference, VDD 두가지 중 하나로 설정 가능

Input range는 다음과 같음



REFERENCE=Internal Reference, GAIN=1/6, single ended mode일 때



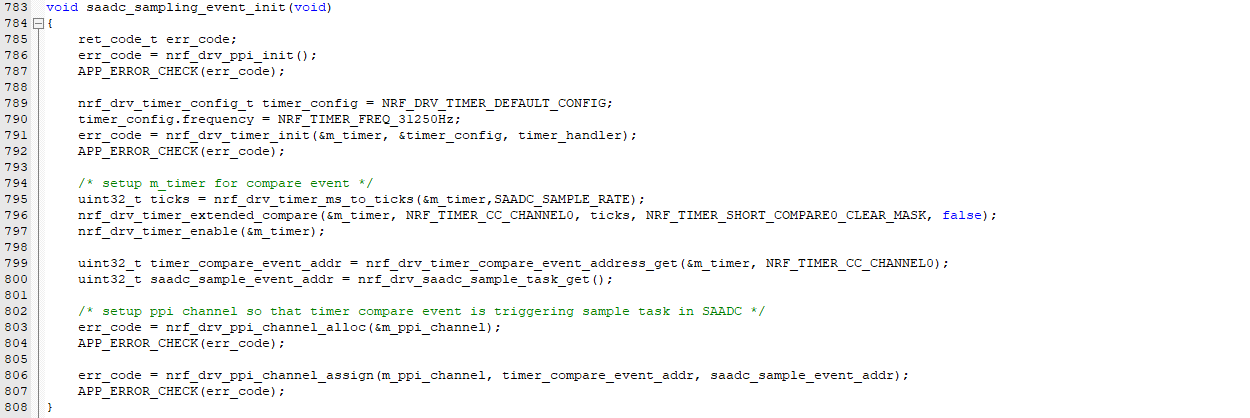
REFERENCE=VDD, GAIN=1/4, single ended mode일 때



**2) 함수 설명**

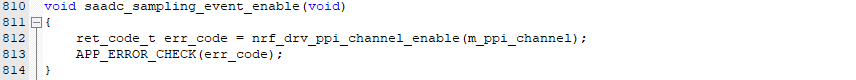
**(1) saadc\_sampling\_event\_init (main.c)**





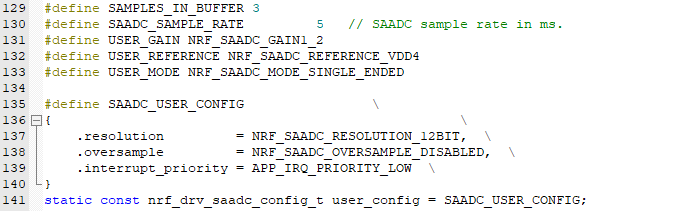
* 타이머, PPI와 같이 adc sampling event에 관여하는 기본값을 초기 설정하는 함수
* [130, 795~797] SAADC\_SAMPLE\_RATE 마다 샘플링을 하도록 타이머 설정
* [803~807] PPI 채널(PPI, Programmable Peripheral Interconnect는 서로 다른 주변 장치가 CPU를 사용하지 않고 Task 및 이벤트를 사용하여 서로 상호 작용할 수 있게 함)을 받아와서 지정

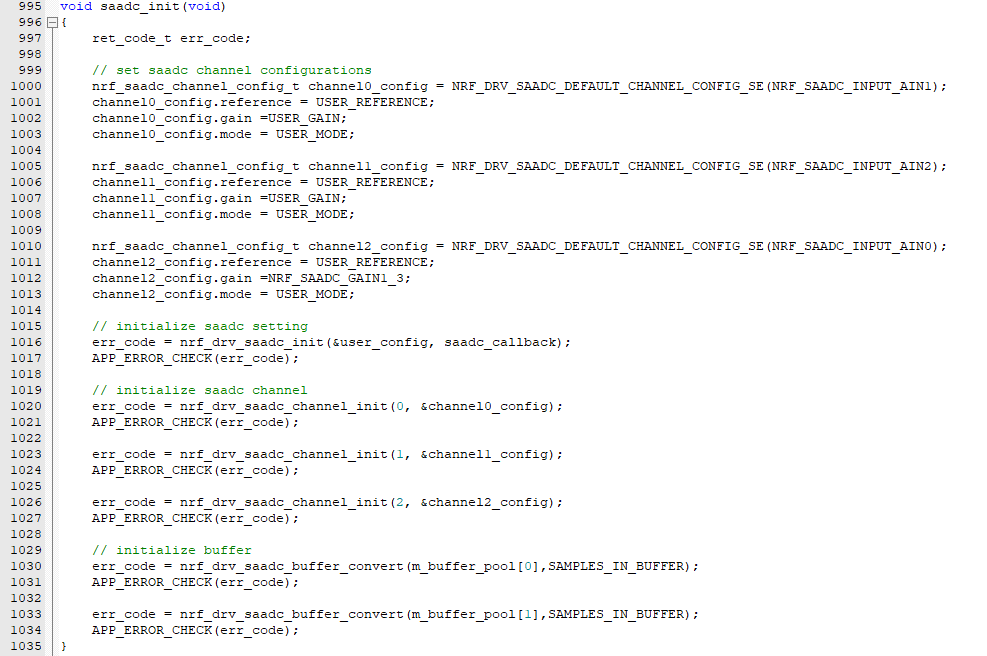
**(2) saadc\_sampling\_event\_enable (main.c)**



* [812~813] PPI 채널을 활성화하는 함수

**(3) saadc\_init (main.c)**





* Reference, resolution, gain 등 adc 세부 설정 값을 초기 설정하는 함수
* [131~133, 1000~1013] 먼저 사용할 Analog Input(AINn) 각각에 saadc channel 설정 값들을 설정
  + Reference=VDD, mode=single ended
  + AIN1, AIN2는 gain이 NRF\_SAADC\_GAIN1\_2(=1/2)인데

AIN0는 gain이 NRF\_SAADC\_GAIN1\_3(=1/3)

=> AIN1, AIN2에는 2.2’ Flex Sensor를 사용하고 AIN0에는 4.5’ Flex Sensor를 사용했는데 센서의 종류에 따라 가장 digital output값이 잘 나오는(센서를 구부림에 따라 최대, 최소값의 차가 가장 큰) gain값을 실험적으로 선택

* [135~141, 1016~1017] saadc 기본값 설정
* Resolution=12bit (가장 정밀한 측정값을 얻기 위해 제일 높은 resolution을 선택)
* Oversampling=disable

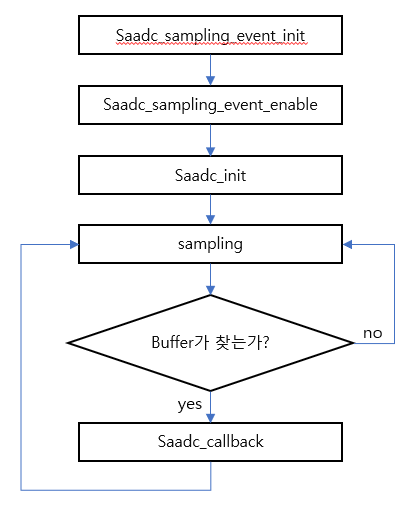
Analog noise를 제거하기 위해 사용하는 방법으로 여러 개의 데이터의 평균 값을 최종 결과값으로 사용, 자체적으로 필터를 사용하였기에 oversampling은 사용하지 않음

* [1020~1027] 설정 값으로 saadc channel 초기화
* [129, 1030~1034] buffer 초기화
* Buffer의 크기가 3인데 (SAMPLES\_IN\_BUFFER=3) 사용하는 analog input이 3개이므로 각각의 input으로부터 매번 값이 들어올 때 마다 buffer가 가득 차고 callback함수가 실행됨

**(4) saadc\_callback (main.c)**

* Adc buffer가 가득차면 실행되는 callback 함수
* MUX, I2C, BLE 관련 함수들을 포함하므로 마지막에 다시 설명

**(5) 동작 사이클**

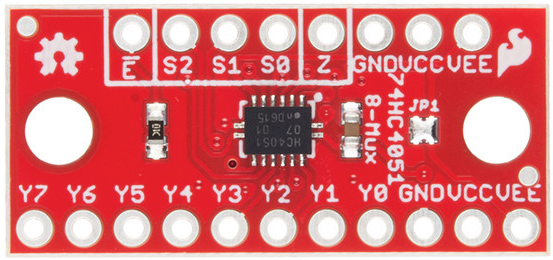


\* 이 때 buffer는 두개가 있는데 한 buffer가 가득 차면 해당 buffer의 값들에 대해 saadc\_callback 함수에서 처리를 하고 새로 들어오는 값들은 다른 buffer에 저장을 한다.

## 4. 스마트 글러브 펌웨어 MUX 코드 설명

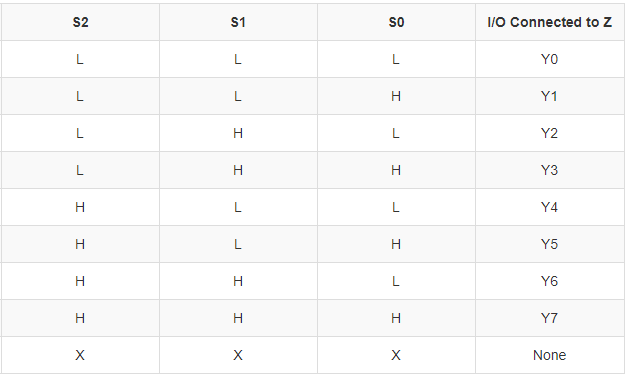
**1) 기본 개념**

**(1) SparkFun Multiplexer Breakout - 8 Channel (74HC4051) (**[**HOOKUP GUIDE**](https://learn.sparkfun.com/tutorials/multiplexer-breakout-hookup-guide?_ga=2.32473594.536085732.1613355437-1014113017.1611991054)**)**

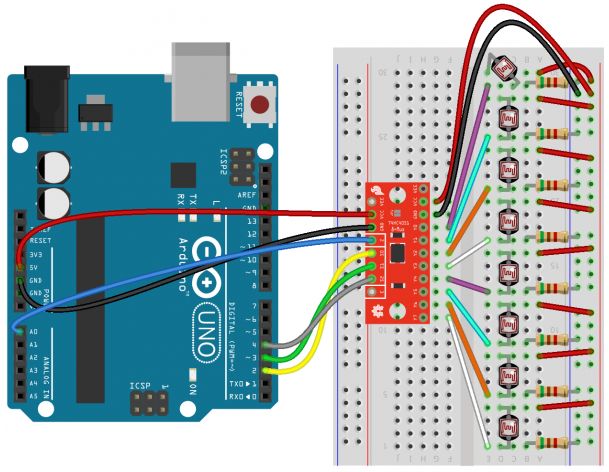


|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Lable** | **Function** | **Input/Output (to board)** | **Notes** |
| S2, S1, S0 | Select Controls | Input | Select inputs |
| Z | Common I/O | Input/Output | Common output or input |
| GND | Ground | Supply | Ground supply voltage |
| VCC | Positive supply | Supply | Positive supply voltage |
| Y7~Y0 | Independent I/O | Input/Output | Selectable I/O to be routed to common pin |

* Select pin (S2~S0)를 통해 Common I/O (Z)에 연결된 Independent I/O pin (Y7~Y0)을 제어할 수 있음



* 만약 S2: L, S1: L, S0: L이면 Y0이 선택되고 Y0으로 들어오는 input 값이 Z를 통해 보드로 전달되거나 (mux가 input으로 사용될 경우) Z를 통해 전달된 output 값이 Y0에 전달됨 (mux가 output으로 사용될 경우)
* **회로도 예시**



* SparkFun VR IMU Breakout - BNO080 Arduino Library를 참고하여 코드 작성

**2) 관련 세팅 값**



* [156] mux의 select pin (S2~S0)에 연결할 보드의 gpio pin

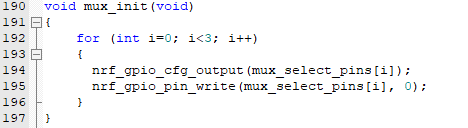
gpio pin 18, 13, 14가 각각 mux의 S0, S1, S2와 연결

* [157] 현재 선택할 mux input
* [158] 최대 mux input의 수

손가락 5개에 각각 2개의 flex sensor가 사용되고 첫 번째 관절을 측정하는 5개의 센서가 하나의 mux에 두 번째 관절을 측정하는 5개의 센서가 또 다른 mux에 연결됨으로 하나의 mux당 input은 최대 5개가 사용됨

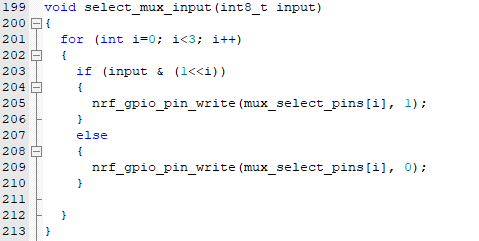
**3) 함수 설명**

**(1) mux\_init (main.c)**



* mux에 사용되는 pin들을 초기 설정하는 함수
* [192~196] 보드의 gpio pin을 output으로 지정해주고 초기 값을 모두 0으로 설정

**(2) select\_mux\_input (main.c)**



* mux input (Y7~Y0)을 선택할 때 사용되는 함수
* Parameters
  + input: 선택할 mux input
    - 선택할 mux input (int8\_t input)에 맞춰서 mux\_select\_pins의 값을 조정

## 5. 스마트 글러브 펌웨어 TWI(I2C) / IMU 코드 설명

**1) 기본개념**

**(1) I2C 통신**

* 다음 문서를 참고

I2C 통신 상세 분석  
[www.hardcopyworld.com/gnuboard5/bbs/board.php?bo\_table=lecture\_pract&wr\_id=95](http://www.hardcopyworld.com/gnuboard5/bbs/board.php?bo_table=lecture_pract&wr_id=95)

**(2) BNO080 (**[**HOOKUP GUIDE**](https://learn.sparkfun.com/tutorials/qwiic-vr-imu-bno080-hookup-guide?_ga=2.2010869.536085732.1613355437-1014113017.1611991054)**)**



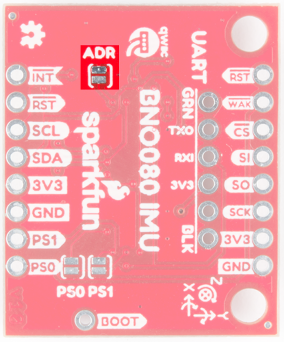
* **Hardware overview**

|  |  |
| --- | --- |
| **Characteristic** | **Range** |
| Operating Voltage | 1.65V – 3.6V |
| Linear Acceleration Accuracy |  |
| Gyroscope Accuracy |  |
| I2C Address | 0x4B or 0x4A |

* **Pins**

|  |  |  |  |
| --- | --- | --- | --- |
| **Pin Lable** | **Function** | **Input/Output (to board)** | **Notes** |
| GND | Ground | Input | 0V/ common voltage |
| 3V3 | Power Supply | Input | Should be between 1.65 – 3.6V |
| SDA | I2C Data Signal | Bi-directional | Bi-directional data line |
| SCL | I2C Clock Signal | Input | Clock signal |

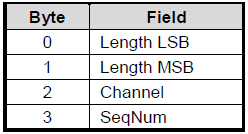
* **I2C ADR jumper**



* I2C ADR jumper를 연결하여 BNO080의 address를 0x4B(default)에서 0x4A로 바꿀 수 있음
* IMU가 손목 회전 측정 용, 어깨 회전 측정 용 총 두 개 사용되기 때문에 하나는 0x4B의 address를 다른 하나는 0x4A의 address를 사용했음
* **SHTP**

BNO080은 Hillcrest의 SHTP (Sensor Hub Transport Protocol)를 사용하여 통신(UART-RVC 제외)

* + 모든 패킷은 4-byte 헤더와 데이터로 구성되어 있고 헤더로 시작됨



* + - Length

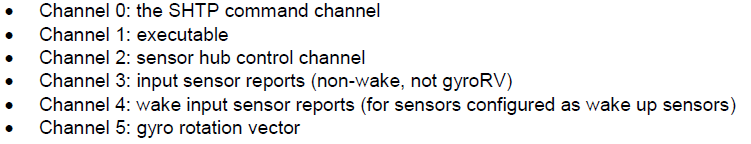
Length field의 15번째 비트는 통신이 이전 통신의 연속인지를 나타냄

14:0 비트는 헤더와 데이터를 합한 총 바이트 수를 나타냄

* + - Channel

데이터의 채널 번호를 나타냄

BNO080에는 6개의 채널이 존재



* + - SeqNum

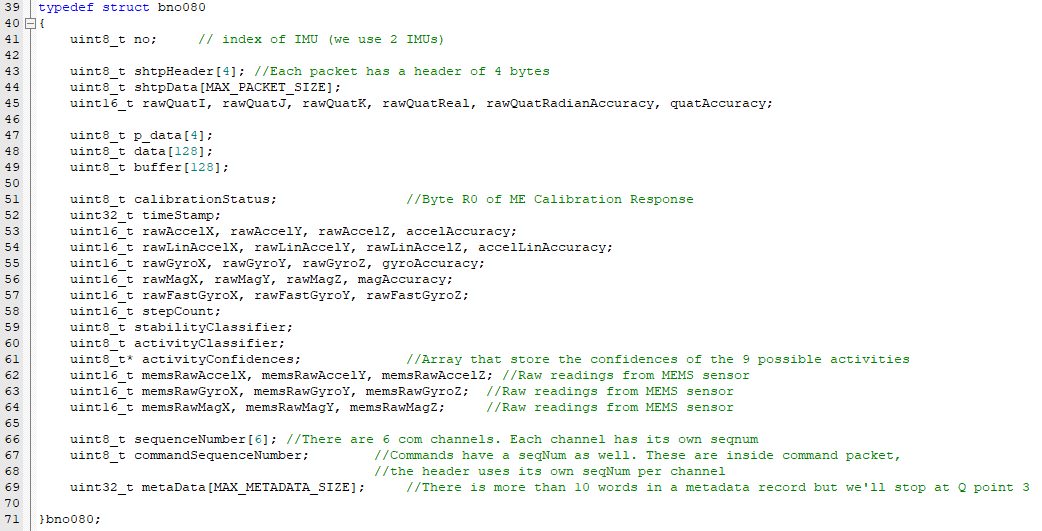
데이터의 순번

전송된 데이터 또는 데이터 연속에 대해 하나씩 증가하는 번호로 각 채널과 방향에는 고유의 SeqNum이 존재

중복 또는 누락된 데이터를 감지하고 분할된 데이터들을 서로 연관시키는데 사용

* SparkFun Multiplexer Breakout - 8 Channel Arduino Library를 참고하여 코드 작성

**2) bno080 구조체**



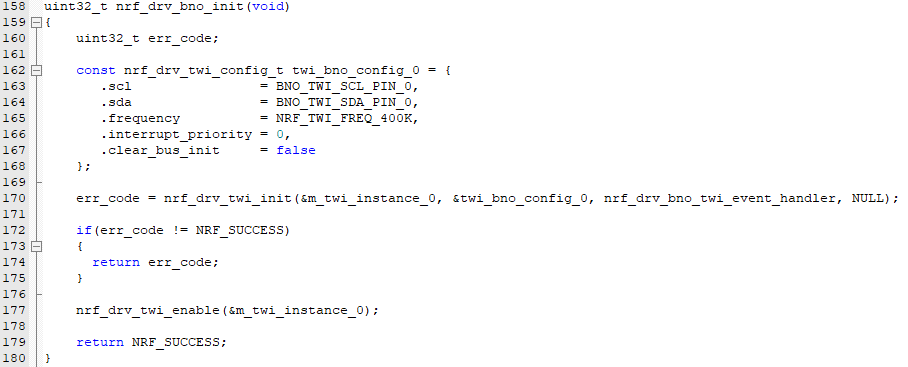
* 각 IMU의 정보를 저장하고 관리하기 위한 구조체
* 멤버
  + [41] no: IMU의 인덱스, 손목과 어깨 총 두 개의 IMU를 사용하므로 각 IMU에 0과 1을 사용
  + [43] shtpHeader: 패킷의 헤더 정보를 저장하기 위한 배열
  + [44] shtpData: 패킷의 데이터 정보를 저장하기 위한 배열
  + [45] rawQuatI, rawQuatJ, rawQuaK, rawQuaReal: 쿼터니언 값을 저장하기 위한 변수
  + [47~49] p\_data, data, buffer: 여러가지 값들을 임시로 저장하는데 사용되는 배열들
  + 나머지 멤버들은 우리 프로젝트에서 크게 사용되지 않는 멤버들이라 설명을 생략 (가속도 센서, 지자기 센서 등 각 센서들의 센서 값이나 캘리브레이션 등의 기능을 위한 기본 멤버들)

**3) 함수 설명**

**(1) nrf\_drv\_bno\_init (bno080.c)**



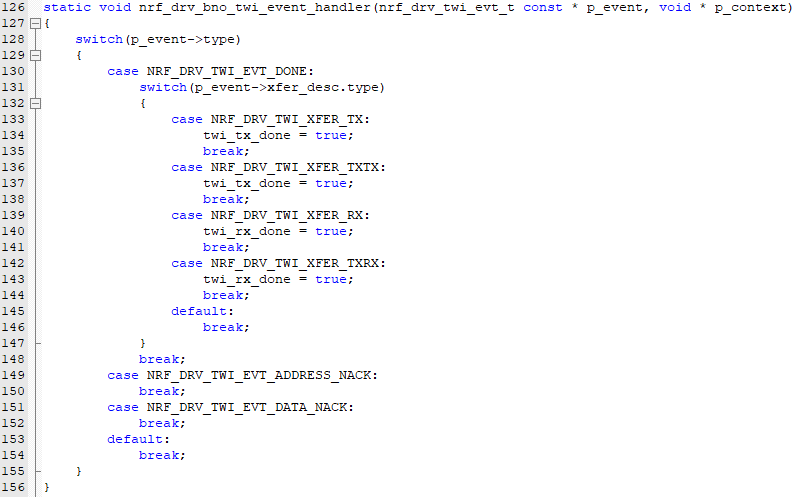




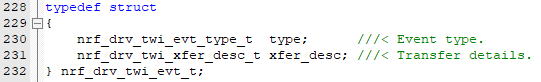
* TWI master 드라이버 인스턴스 초기화 및 활성화를 위한 함수
* [162~168] TWI master 드라이버 인스턴스 구성을 위한 구조체(nrf\_drv\_twi\_config\_t) 설정
  + [79] scl: SCL pin number, 15번을 사용
  + [80] sda: SDA pin number, 16번을 사용
  + frequency: TWI master clock frequency, 400kbps로 설정
* [170] TWI master driver instance 초기화
  + &m\_twi\_instance\_0: 드라이버 인스턴스 구조체 포인터
  + &twi\_bno\_config\_0: 초기 구성을 위한 구조체 포인터
  + nrf\_drv\_bno\_twi\_event\_handler: TWI 이벤트 핸들러
* [177] TWI 인스턴스 활성화

**(2) nrf\_drv\_bno\_twi\_event\_handler (bno080.c)**

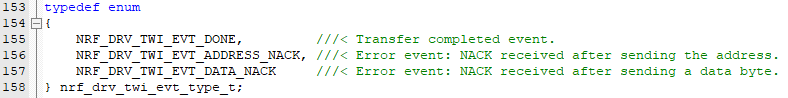




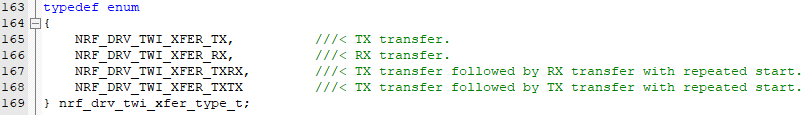
* TWI 이벤트가 발생했을 때 실행되는 함수
* Parameters
  + p\_event: TWI 이벤트 구조체



* type: TWI 이벤트 종류를 나타내는 enumeration

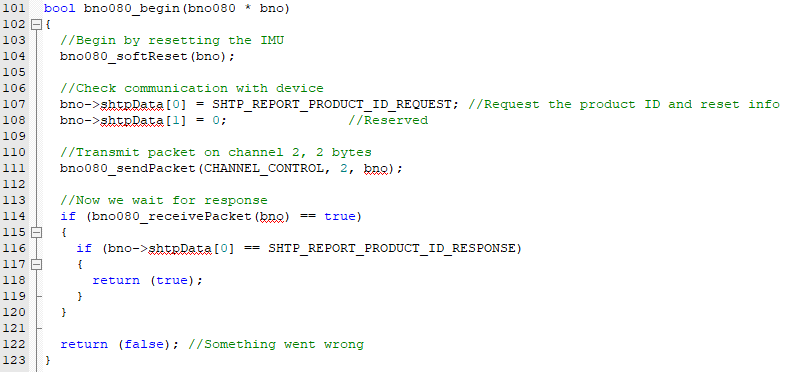


* + - NRF\_DRV\_TWI\_EVT\_DONE: 통신 완료
    - NRF\_DRV\_TWI\_EVT\_ADDRESS\_NACK: address를 전송 후 에러 발생
    - NRF\_DRV\_TWI\_DATA\_NACK: data를 전송 후 에러 발생
  + xfer\_desc: TWI 통신 종류를 나타내는 enumeration



* + - * NRF\_DRV\_TWI\_XFER\_TX: 쓰기 작업
      * NRF\_DRV\_TWI\_XFER\_TXTX: 쓰기 작업 후 쓰기 작업
      * NRF\_DRV\_TWI\_XFER\_RX: 읽기 작업
      * NRF\_DRV\_TWI\_XFER\_TXRX: 쓰기 작업 후 읽기 작업
    - p\_context: 이벤트 핸들러에 전달되는 context (본 시스템에선 사용하지 않음)
      * [134] 통신이 정상적으로 이루어 졌고 통신 종류가 TX라면 twi\_tx\_done=true
      * [137] 통신이 정상적으로 이루어 졌고 통신 종류가 TXTX라면 twi\_tx\_done=true
      * [140] 통신이 정상적으로 이루어 졌고 통신 종류가 RX라면 twi\_rx\_done=true
      * [143] 통신이 정상적으로 이루어 졌고 통신 종류가 TXRX라면 twi\_rx\_done=true
      * [149] 통신 중 address를 전송 후 에러 발생
      * [151] 통신 중 data를 전송 후 에러 발생

**(3) bno080\_begin (bno080.c)**

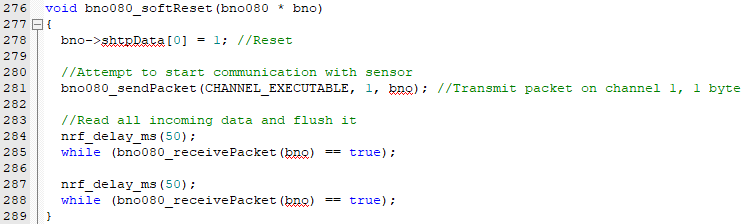


* IMU(bno080)를 처음 동작 시키는 함수
* Parameters
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* IMU를 소프트웨어적으로 리셋[104]
* [107~111] product ID를 요청하는 2byte 패킷을 2번 채널로 전송
  + CHANNEL\_CONTROL이 2번 채널



* [114~122] IMU로부터 전송된 패킷을 받아서 product ID 요청이 제대로 이루어졌는지 확인
  + 모든 작업이 완료되었다면 true를 반환 그렇지 않고 에러가 발생했다면 false를 반환

**(4) bno080\_softReset (bno080.c)**

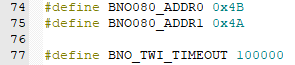


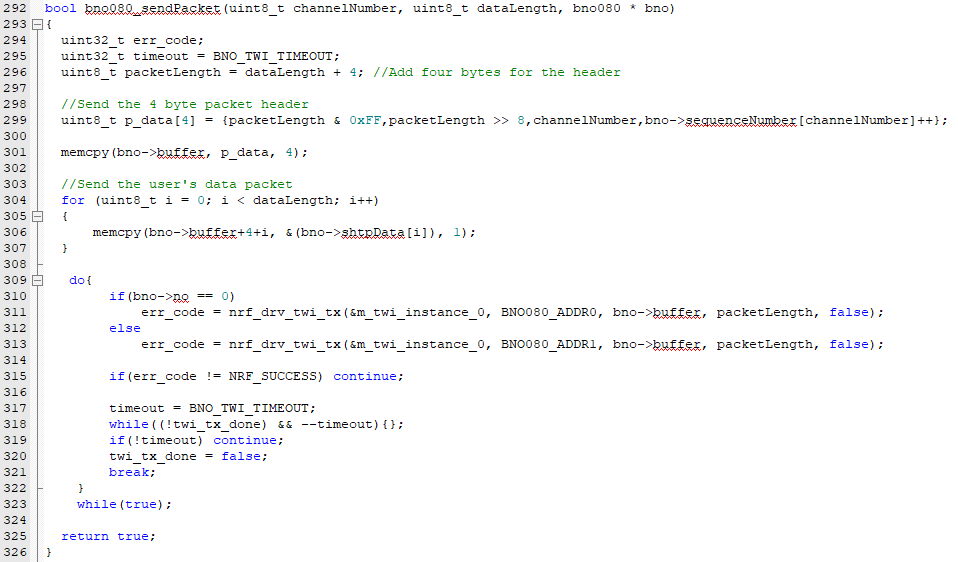
* IMU를 소프트웨어적으로 리셋 시키는 함수
* Parameters
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [278~281] Reset을 요청하는 1byte 패킷을 1번 채널로 전송
  + CHANNEL\_EXECUTABLE이 2번 채널



* [284~288] 들어오는 모든 데이터를 읽어서 비움

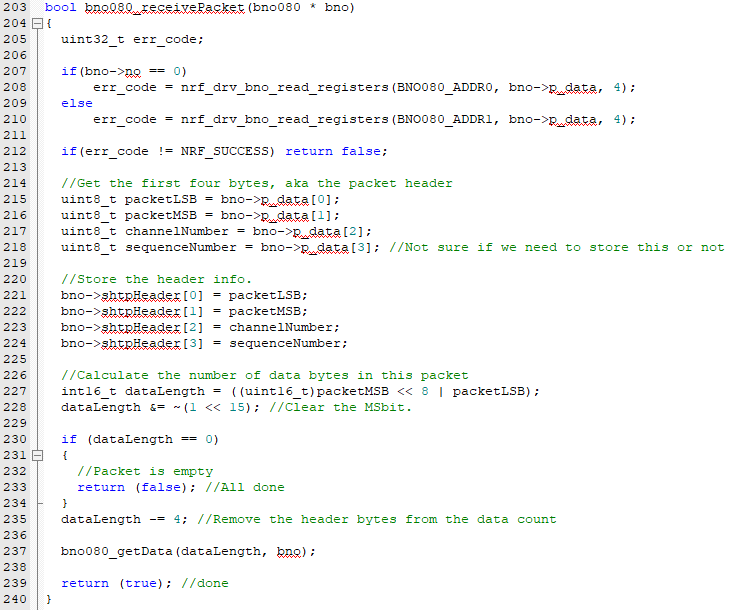
**(5) bno080\_sendPacket (bno080.c)**





* IMU의 x번(channelNumber) 채널에 y(dataLength) byte 데이터를 가진 패킷을 전송하는 함수
* Parameters
  + channelNumber: 패킷을 전송할 채널
  + dataLength: 전송할 데이터 길이(byte)
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [296] 전송할 패킷의 총 길이는 데이터 길이(dataLength) + 헤더의 길이(4)
* [299~301] bno080 구조체의 buffer에 헤더를 복사
* [304~307] bno080 구조체의 buffer에 데이터를 복사
* [309~323] IMU에 패킷을 전송
  + [311] bno080구조체의 no가 0이라면 BNO080\_ADD0의 주소 값을 가지는 slave(IMU)로 전송
  + [313] bno080구조체의 no가 1이라면 BNO080\_ADD1의 주소 값을 가지는 slave(IMU)로 전송
  + [315~321] BNO\_TWI\_TIMEOUT의 반복 횟수의 while문 안에 통신이 성공적으로 이루어 지지 않으면 전송 작업을 반복

**(6) bno080\_receivePacket (bno080.c)**



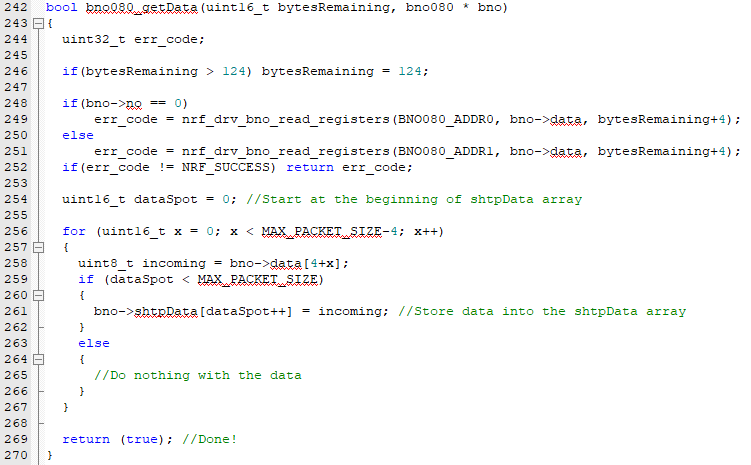
* IMU로부터 패킷을 읽어 오는 함수
* Parameters
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [210~213] bno080 구조체의 no에 따라 해당 IMU의 데이터를 4byte 만큼 읽어 옴
  + 패킷 헤더를 읽어 오는 과정
* [218~227] 읽어온 패킷 헤더를 bno080 구조체의 shtpHeader 배열에 저장
* [230~238] 패킷 헤더의 정보를 통해 패킷 데이터의 길이를 계산
* [240] 패킷 데이터의 길이만큼 데이터를 읽어 옴

**(7) nrf\_drv\_bno\_read\_registers (bno080.c)**



* addr 주소 값을 가지는 IMU의 데이터를 length byte 만큼 읽어서 p\_data 위치에 저장하는 함수
* Parameters
  + addr: slave(IMU) 주소
  + p\_data: 읽어 들인 데이터를 저장할 버퍼
  + length: 읽어 올 데이터의 길이
* [188] TWI slave로부터 데이터를 읽어 옴
* [187~198] BNO\_TWI\_TIMEOUT의 반복 횟수의 while문 안에 통신이 성공적으로 이루어 지지 않으면 작업을 반복

**(8) bno080\_getData (bno080.c)**



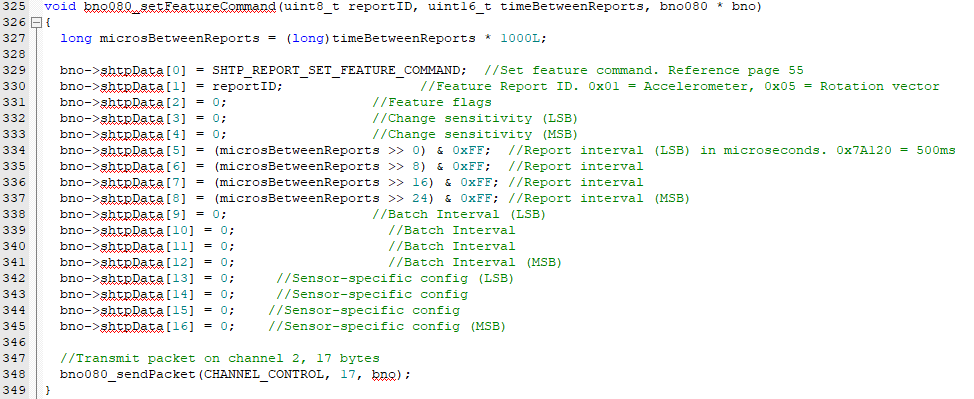
* IMU 패킷의 데이터 부분을 읽어 오는 함수
* Parameters
  + bytesRemaining: 패킷 데이터의 길이
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [246~252] bno080 구조체의 no에 따라 해당 IMU의 데이터를 bytesRemaining+4 byte 만큼 읽어 옴
  + IMU의 데이터는 헤더부터 시작하기 때문에 bytesRemaining에 헤더의 길이인 4를 더한 길이의 데이터를 읽어 옴. 즉, 패킷 전체를 읽어 오는 과정
* [256~267] 읽어온 패킷 데이터를 bno080 구조체의 shtpData 배열에 저장

**(9) bno080\_enableRotationVector (bno080.c)**



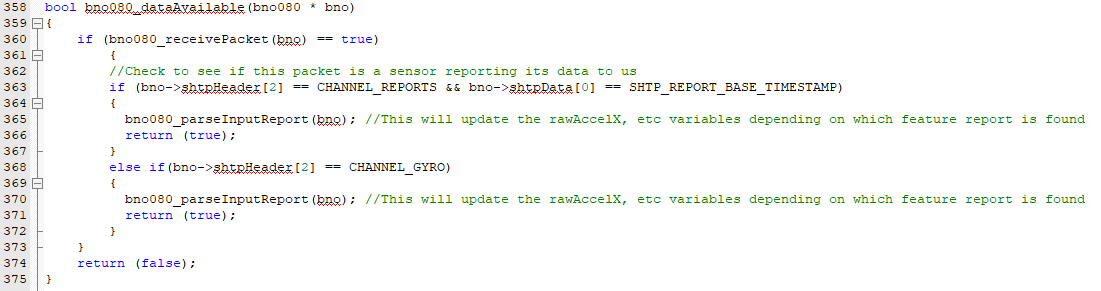
* IMU가 매 timeBetweenReports(ms) 초마다 쿼터니언 값을 전송할 수 있게 활성화시키는 함수
* Parameters
  + timeBetweenReports: 동작 주기 (ms)
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* IMU가 매 timeBetweenReports(ms) 초마다 쿼터니언 값을 전송할 수 있게 활성화

**(10) bno080\_setFeatureCommand (bno080.c)**



* IMU의 여러가지 센서 및 기능을 활성화시키기 위한 함수
* Parameter
  + reportID: 활성화시키고자 하는 기능의 종류
  + timeBetweenReports: 동작 주기 (ms)
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [327~348] reportID 기능이 timeBetwwenReports 주기 마다 동작하게 활성화한다는 정보가 담긴 패킷을 IMU에 전송

**(11) bno080\_dataAvailable (bno080.c)**



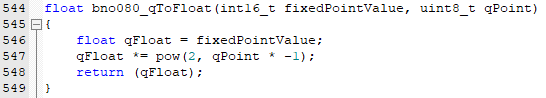
* IMU에 읽어올 수 있는 새로운 데이터가 있는지 확인하는 함수
* saadc\_callback 함수에서 실행되며 adc 데이터 (flex sesor 값)을 전달할 때 실행되어 IMU 데이터를 전달하기 위해 사용됨
* Parameters
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* [360~379] IMU에서 패킷을 읽어 오는 것이 가능하면 해당 패킷의 정보에 맞는 동작을 실행하고 true를 반환, 패킷을 읽는 것이 불가능하면 false를 반환
  + [363~367] 현재 채널이 CHANNEL\_REPORTS이고 report ID가 SHTP\_REPORT\_BASE\_TIMESTAMP라면 IMU로부터 데이터를 가져와서 저장
    - CHANNEL\_REPORTS는 input sensor report 채널로 단방향이며 IMU에서 호스트로 데이터를 전달
    - SHTP\_REPORT\_BASE\_TIMESTAMP는 timestamping을 사용한다는 것을 의미하며 호스트 애플리케이션에서 정확한 타임 스탬프를 작성할 수 있음
  + 현재 채널이 CHANNEL\_GYRO이라면 IMU로부터 데이터를 가져와서 저장[368~372]
    - CHANNEL\_GYRO는 gyro rotation vector 채널로 자이로 회전 벡터에 사용되는 채널. 낮은 지연 시간을 위해 별도의 채널이 제공

**(12) bno080\_parseInputReport (bno080.c)**



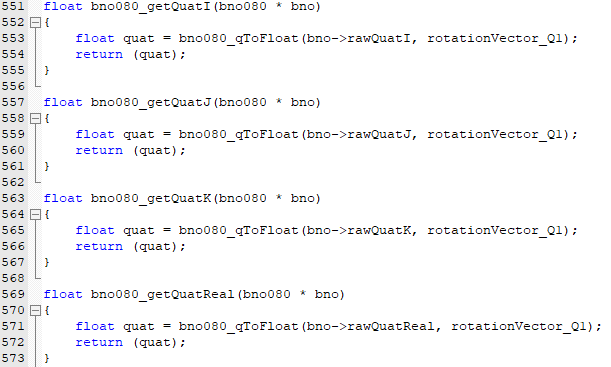
* IMU 데이터로부터 원하는 정보 값을 계산하여 적절한 변수에 저장하는 함수
* Parameters
  + bno: bno080 구조체 포인터, 현재 동작 시키고자 하는 IMU를 의미
* IMU의 데이터로부터 원하는 정보 값을 계산[413~418]
* [431~441] 현재 report ID 가 rotation vector라면 위에서 계산 한 값이 쿼터니언 값이라는 것을 의미하므로 이를 알맞은 변수들에 저장
  + rotation vector를 제외한 나머지 기능에 관한 경우는 우리 시스템에서 사용하지 않으므로 나머지 부분은 생략

**(13) bno080\_qToFloat (bno080.c)**



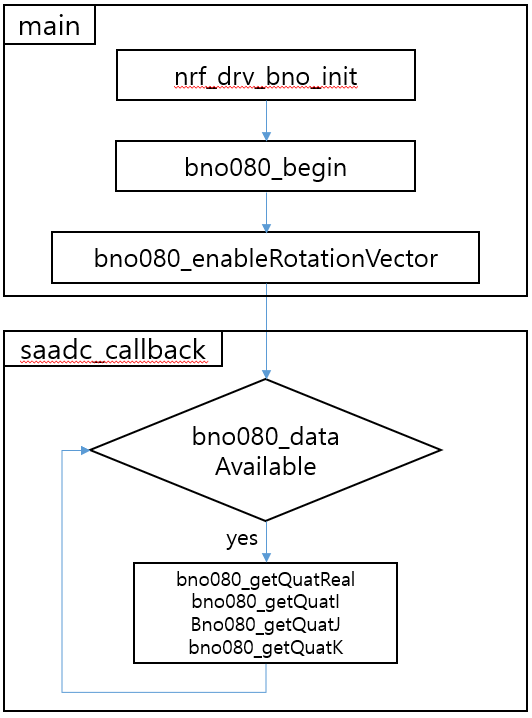
* 쿼터니언 값이 주어지면 fixed point floating을 regular floating point로 변환하는 함수

**(14) bno080\_getQuatI, bno080\_getQuatJ, bno080\_getQuatK, bno080\_getQuatReal (bno080.c)**



* 쿼터니언 값들을 반환하는 함수
* saadc\_callback 함수에서 실행되며 adc 데이터 (flex sesor 값)을 전달할 때 실행되어 IMU 데이터를 전달하기 위해 사용됨

**(15) 동작 사이클**

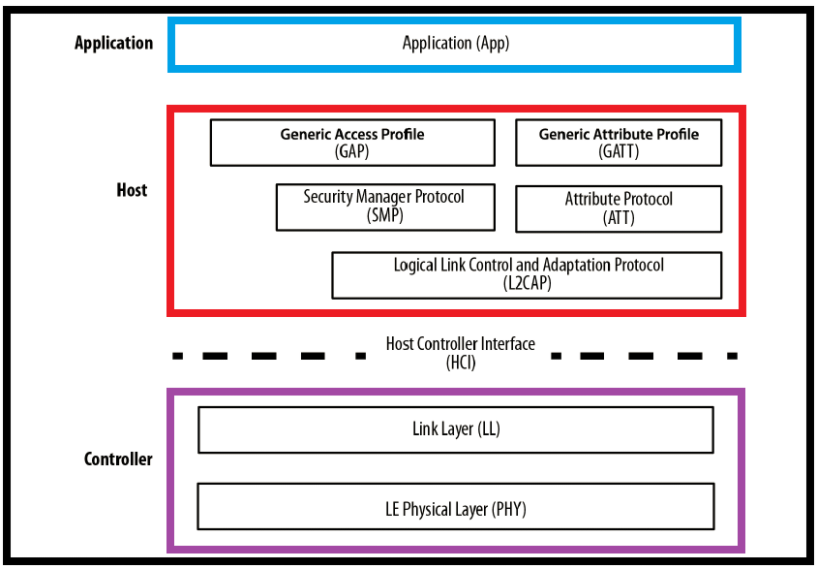


## 6. 스마트 글러브 펌웨어 BLE 코드 설명

**1) 기본 개념**

**(1) BLE (출처:** <http://www.hardcopyworld.com/gnuboard5/bbs/board.php?bo_table=lecture_iot&wr_id=11>)

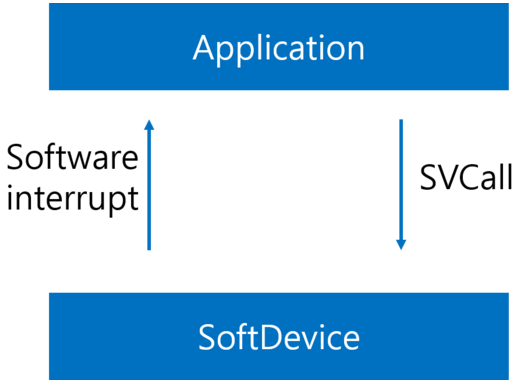
* **Protocol Stack**



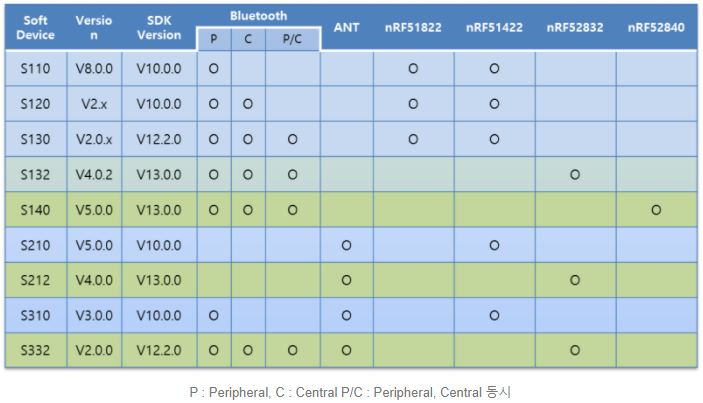
* + Application-Host-Controller 3개의 계층 구조를 이루며 동작
* **GAP(Generic Access Profile)**
  + 다양한 제조사에서 제작된 BLE 장치들이 서로 연동될 수 있도록 프레임워크를 제공
* **GATT(GENERIC ATTRIBUTE PROFILE)**
  + BLE 에서는 어떤 장치와 연결되더라도 일관된 방식으로 데이터를 주고받을 수 있도록 데이터 형식(format)과 동작방식(procedure)을 추상화
  + BLE 표준은 시장에서 자주 사용될 수 있는 장치들과 그 속에 포함될 기능, 데이터 들을 미리 정의해 두었는데 이를 Profile이라 부름, 이런 Profile 들이 실제 BLE 장치속에서 구현될 때 갖춰야 할 데이터 형식, 데이터의 계층 구조, 동작방식이 GATT를 기반으로 함
* 자세한 내용은 출처 참고

**(2) SoftDevice (출처:** <https://blog.naver.com/htm2937/220748533243>**)**

* Nordic사에서 제공하는 BLE 통신 관련 기능들을 처리하기 위한 프로토콜 스택
* SoftDevice는 사용자의 어플리케이션에서 값을 입력한대로 설정되어 외부로부터 들어오는 통신 신호를 받아 처리해서 이를 사용자에게 소프트웨어 인터럽트 신호를 통해 전달. 사용자는 인터럽트를 받아 처리가 필요한 ble 이벤트가 있는지 확인하고, 있다면 해당 동작을 처리하도록 미리 등록한 이벤트 핸들러를 실행시키는 구조
* 즉 물리 계층에 가까운 동작은 SoftDevice 측에서 다 처리되고 어플리케이션에서는 받은 패킷 데이터를 원하는 대로 처리하면 됨



* SoftDevice에는 다양한 버전이 존재하는 데 우리가 사용하는 nRF52832 칩과 호환되는 버전인 S132를 사용

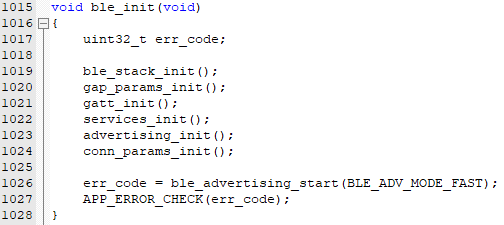


**(3) NUS (Nordic UART Service)**

* BLE를 통해 데이터를 송수신하여 UART로 전달할 수 있게 하는 서비스

**2) 함수 설명**

**(1) ble\_init (main.c)**



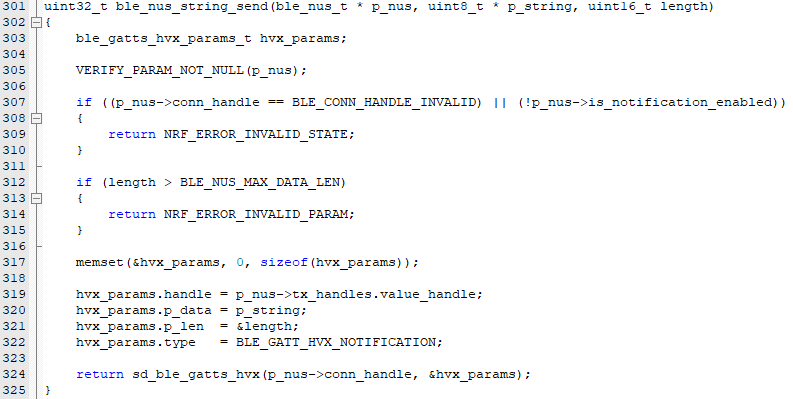
* BLE 관련 초기 설정을 해주는 함수
* [1019] bke\_stack\_init
  + soft device와 ble 관련 초기화를 하는 함수
  + soft device 초기화/활성화, ble 최대 연결 수 등을 설정
* [1020] gap\_parmas\_init
  + GAP(Generic Access Profile) 관련 초기화를 하는 함수
  + 디바이스의 GAP parameter들을 설정, connection interval(master와 slave가 데이터를 주고받는 시간 간격)과 같은 연결 관련 parameter, 디바이스 이름 등을 포함
* [1021] gatt\_init
  + GATT(GENERIC ATTRIBUTE PROFILE) 관련 초기화를 하는 함수
  + MTU(maximum transmission unit)값 등을 설정
  + MTU값은 원래 20(byte)로 설정되어 있는데 우리가 전송하고자 하는 데이터의 크기는 39byte이므로 이 값을 바꾸어 주어야 함



여기서 BLE\_GATT\_ATT\_MTU\_DEFAULT를 43으로 설정해 줬음

* [1022] services\_init
  + 사용할 서비스 관련 초기화를 하는 함수
  + 우리 프로그램에선 NUS를 사용
* [1023] advertising\_init
  + advertising 관련 초기화를 하는 함수
* [1024] conn\_params\_init
  + 최대 연결 시도 횟수 등 Connection Parameters Negotiation 관련 초기화를 하는 함수

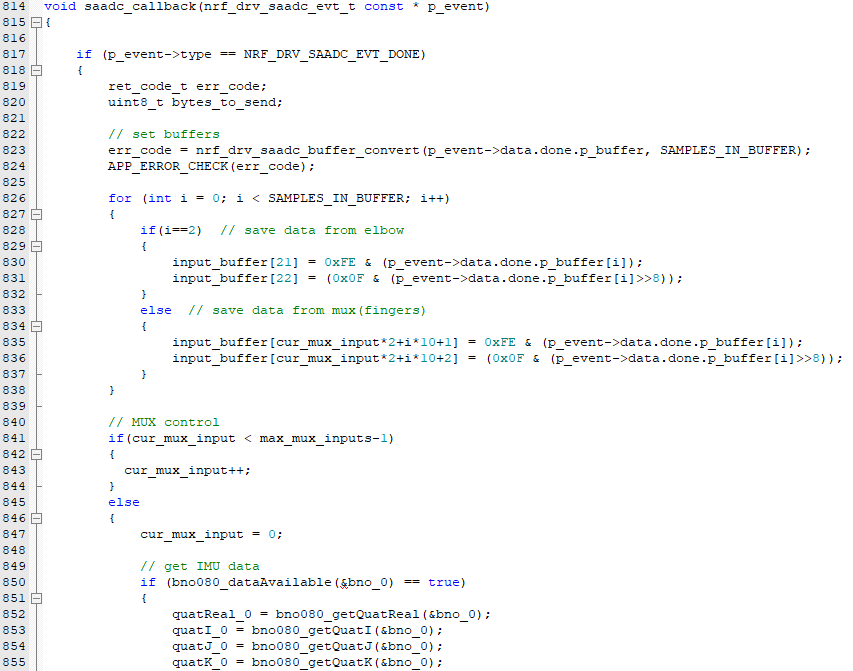
**(2) ble\_nus\_string\_send (main.c)**

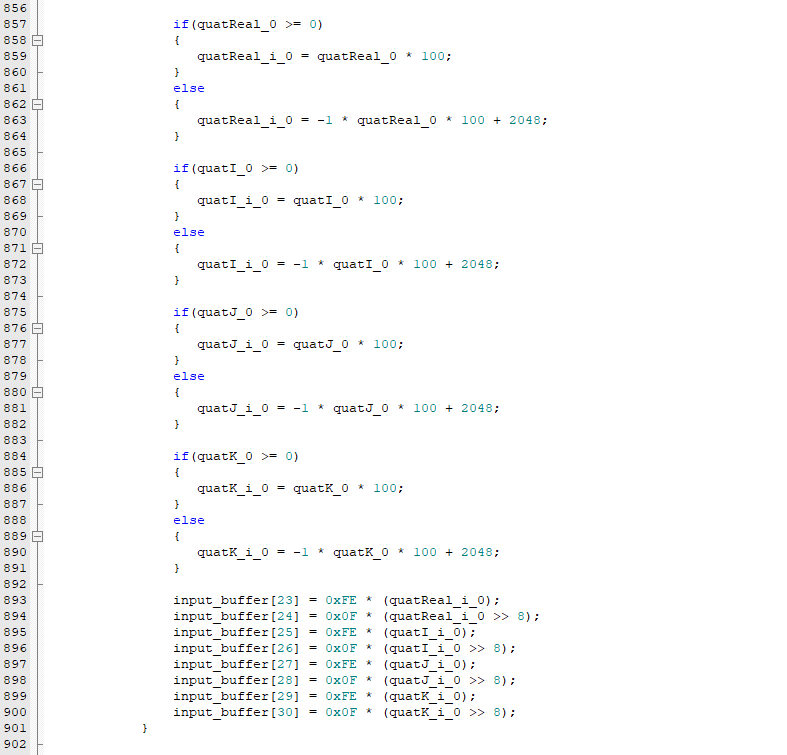


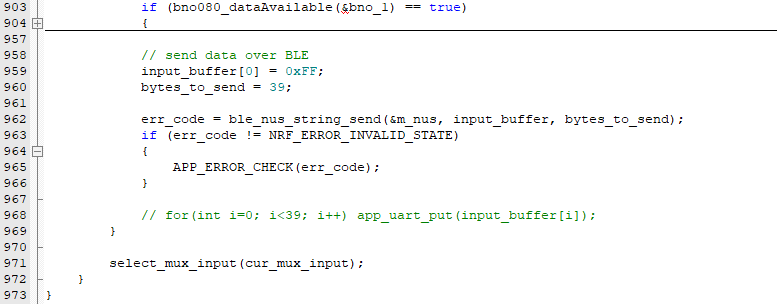
* 문자열을 연결된 상대 디바이스로 전송하는 함수
* Parameters
  + p\_nus: NUS 구조체 포인터
  + p\_sting: 전송하고자 하는 문자열
  + length: 전송하고자 하는 문자열의 크기
* SAADC의 saadc\_callback 함수 안에서 실행되면서 모든 센서의 값들이 측정되면 측정된 값을 전달

## 7. 스마트 글러브 펌웨어 전체 구성

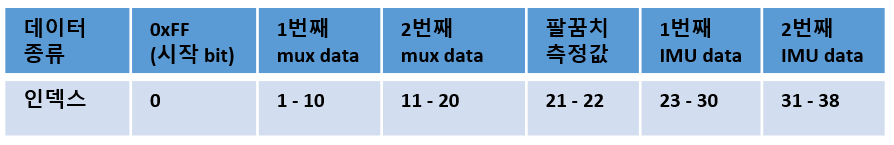
**1) saadc\_callback (main.c)**







* Adc buffer가 가득차면 실행되는 callback 함수, BLE를 통한 측정 값의 전송을 포함
* Prameters
  + - * p\_event: adc driver event 구조체, adc 측정 값, 크기 등을 포함
    - 데이터 패킷 구조



* 시작 bit는 PC프로그램에서 패킷의 시작을 알 수 있게 하기 위한 용도
  + - [826~838] adc buffer안에 있는 측정값을 최종 측정값 저장 배열인 input\_buffer에 저장
      * [828-832] adc buffer의 2번째 값은 항상 팔꿈치 측정값이고 이 값을 input\_buffer의 21,22번째 값으로 저장
      * [833-837] adc buffer의 0,1번째 값은 손가락 측정값인데 현재 측정한 mux의 input(cur\_mux\_input)값에 따라 알맞은 위치에 저장
      * 이때 input\_buffer는 8bit int형 배열인데 측정한 adc값은 12bit이므로 12bit 값을 8bit, 4bit로 나누어서 저장
        + 예를 들어 adc값이 [0101 1100 0011]이라면 [0000 0101], [1100 0011]로 나누어서 저장
        + 이때 상위 8bit에는 0x0F[0000 1111] 하위 8bit에는 0xFE를 곱해주는데 이는 PC에서 수신 받은 패킷의 시작을 확인하기 위한 용도로 input\_buffer의 0번째 값을 0xFF로 지정하고 이 값을 받은 후부터 패킷을 데이터를 분석하기 시작하므로 위와 같은 전 처리 작업을 시행
    - [841~844] 만약 측정해야 할 mux값이 남아 있다면 cur\_mux\_input을 1 증가시켜서 mux의 다음 input을 측정
    - [845~969] mux 값을 모두 측정했다면 IMU의 값을 받아오고 BLE로 전송
* [847] cur\_mux\_input을 다시 0으로 초기화
  + - [850~901] 0번 IMU(bno\_0)의 값을 받아와서 input\_buffer에 저장
      * [852~855] IMU의 쿼터니언 값(float 형)을 저장
      * [857~891] 쿼터니언 값을 가공
        + 만약 양수라면 100을 곱하고 int값으로 저장하여 float값의 소수점 2번째 자리 까지를 저장, ble로 데이터를 전송할 때 사용하는 ble\_nus\_string\_send함수가 unsined 8bit int형만을 지원하므로 해당 작업을 시행, 이후 PC에서 수신 받은 데이터를 100으로 나누어서 원래 값을 복원

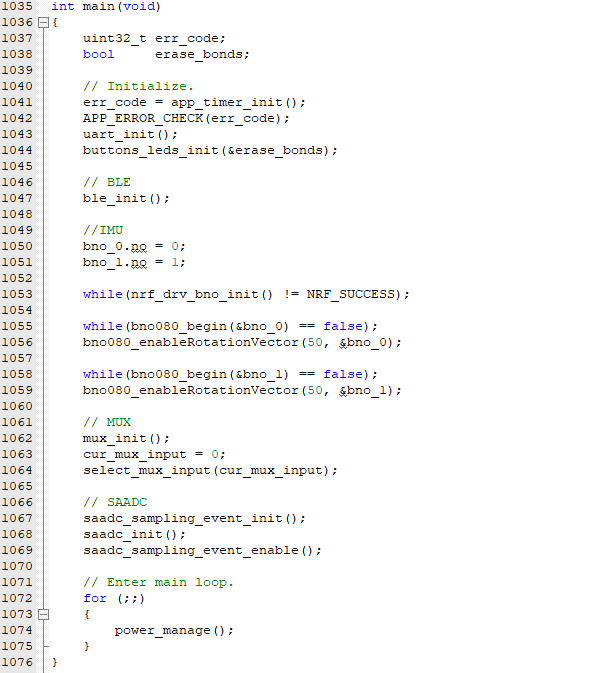
예를 들어 쿼터니언 값(float 형)이 0.4543… 이라면 100을 곱한 값인 45를 int형으로 저장하고 이 값을 BLE로 PC에 전송하면 PC 프로그램에선 45를 100으로 나누어 0.45를 쿼터니언 값으로 사용

* + - * + 만약 음수라면 -1을 곱하여 양수로 바꾼 후 100을 곱하고 2048을 더한 int값으로 저장, 이 때 양수로 바꾸고 2048을 더해주는 이유는 unity에서 시리얼 데이터를 받아오는 함수가 8bit 시리얼 데이터를 4byte로 저장하는데 형 변환 과정에서 일어나는 오류를 피하기 위해 위와 같은 작업을 시행

예를 들어 쿼터니언 값(float 형)이 -0.4543… 이라면 45+2048=2093을 전송하고 PC 프로그램에서 이 값에 2048을 빼고 -1을 곱한 값인 -45를 100으로 나눈 -0.45를 쿼터니언 값으로 사용

* + - * [893~900] 쿼터니언 값을 input\_buffer에 저장
    - [903~957] 1번 IMU(bno\_0)의 값을 받아와서 input\_buffer에 저장, 반복되는 코드이므로 자세한 설명은 생략
    - [959] input\_buffer의 0번째 값을 0xFF로 지정
    - [962~966] 모든 측정값들을 BLE로 전송

**2) main (main.c)**



* + main 함수에선 각종 기능 관련 초기화 및 활성화를 실행
  + [1047] BLE 초기화 및 활성화
  + [1050~1059] I2C/IMU 초기화 및 활성화
  + [1062~1064] MUX 초기화
  + [1067~1069] ADC 초기화 및 활성화

**3) 전체 동작 사이클**

