

Custom STM32F429ZI 부트로더 만들기

제작자: 유영재

목차

1.	부트로더 개요.....	3
1.1	Q&A를 통한 부트로더 예시.....	3
1.2	Memory 관리.....	4
1.3	Keil을 통한 Memory View	6
1.4	Boot Configuration	7
1.5	개발환경 구성	7
1.6	Keil 프로젝트 분석.....	14
1.7	Led blink 파일 작성.....	17
1.8	OpenSTM32 System Workbench 개발환경 구축	23
2.	System Memory 부트로더.....	28
2.1	ST Native Bootloader	29
2.2	Flash loader demonstrator 설치 및 실행	30
2.3	다른 UART로 시스템 부트로더 사용하기.....	34
2.4	더 나아가기. Hex 파일 이해하기	35
2.5	시스템 부트로더 Flowchart.....	36
3.	Custom 부트로더 작성	45
3.1	Block Diagram	45
4.	PC Host 통신 프로그램 작성 [Python].....	48
4.1	python 설치.....	48
4.2	main 작성.....	50
4.3	decode command.....	52
5.	USB DFU 부트로더 작성	59
5.1	USB 기본 개념	59
5.2	USB Key를 이용한 펌웨어 업데이트	60
5.3	다른 방법으로 DFU 구현하기.....	72

1. 부트로더 개요

- 부트로더는 단순하게 생각하면 작은 코드 덩어리라고 말할 수 있고, 실제로도 소프트웨어의 일부분으로 동작한다. STM Chip에서는 MCU ROM에 내장 부트로더가 있고 혹은 FLASH, SRAM에 사용자가 임의로 만들 수 있게 되어 있다
- 사전에 MCU 프로그램을 하다 보면, IAP(In-Application Programming)과 ISP(In-System Programming) 용어를 많이 접할 수 있다. 간단하게 생각하면, IAP와 ISP 모두 MCU에 프로그램을 업로드하기 위한 방식인데, 둘 사이의 정확한 차이를 알아야 한다. ARM에서는 IAP와 ISP를 아래와 같이 정의하고 있다

In-System Programming means that the device can be programmed in the circuit by using an utility such as the ULINK USB-JTAG Adapter.

In-Application Programming means that the application itself can re-program the on-chip Flash ROM.

결론적으로 ISP는 ULINK USB-JTAG 장비를 포함하여 Serial 및 특정 인터페이스를 이용하여 MCU에 프로그램 하는 방식을 일컫는다. 그리고 IAP는 Application 자체 내에서 Flash ROM에 재 프로그래밍을 하는 것이다

- 현재 우리가 하려는 프로젝트는 IAP 개발과 연관되었다고 말할 수 있다. 만드려는 부트로더의 주 역할은 Application Loader라고 말할 수 있다. 이를 포함해서 원할 때만 Application을 업데이트하는 메커니즘을 가지고 있다

1.1 Q&A를 통한 부트로더 예시

A. Arduino Uno

- a. 해당 보드는 부트로더를 가지고 있는가?
그렇다. Chip 안에 부트로더가 내장되어 있다
- b. MCU가 reset 때마다 동작을 하는가?
그렇다. 리셋 후에 바로 내장 부트로더가 동작을 한다
- c. 해당 보드에서의 부트로더의 주 역할은?
주로 Arduino 펌웨어(혹은 Sketch)를 다운로드 하는데 쓰인다

B. STM Nucleo 64

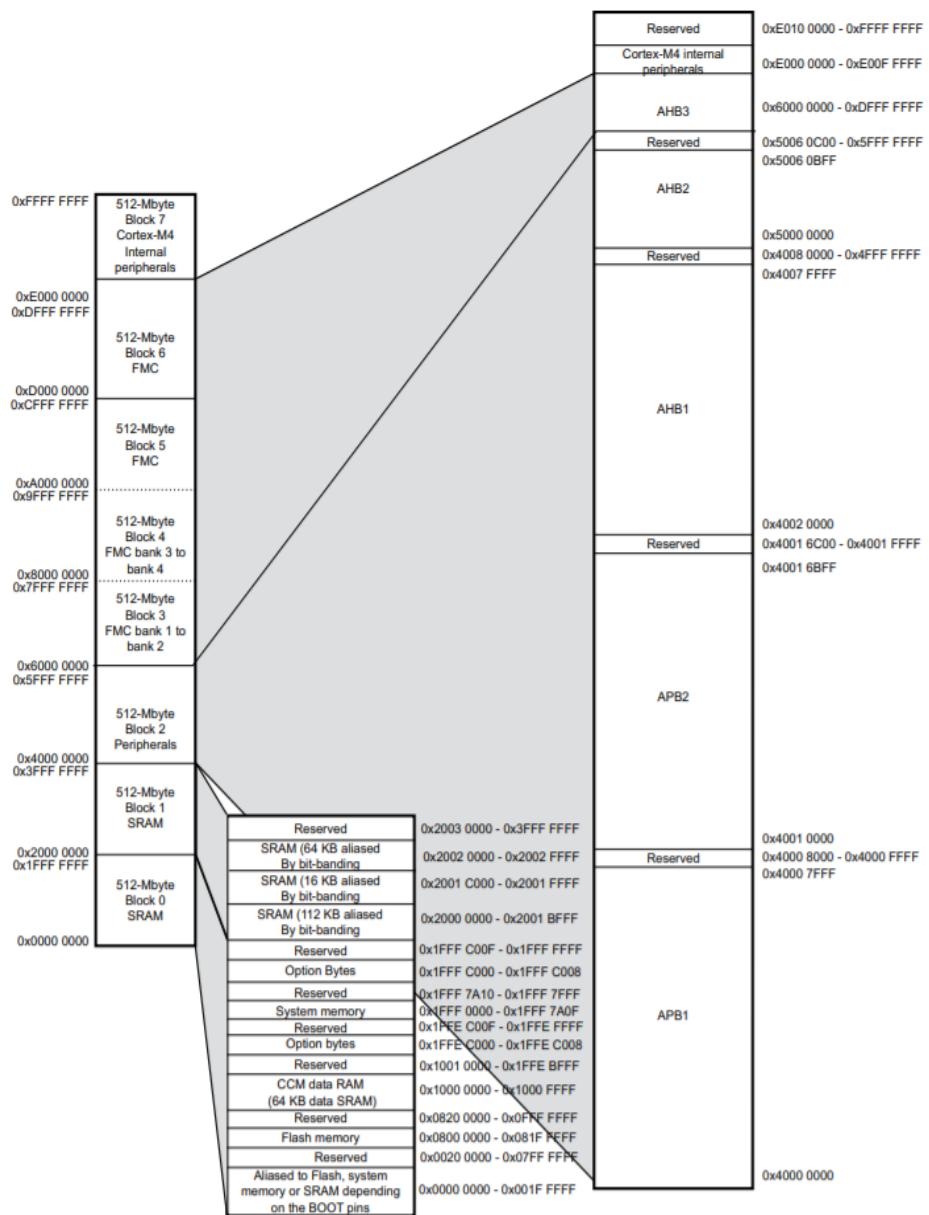
- a. 해당 보드는 부트로더를 가지고 있는가?
그렇다. Chip 안에 부트로더가 내장되어 있다
- b. MCU가 reset 때마다 동작을 하는가?
아니다. 기본적으로는 동작하지 않고, Boot Pin의 상태를 바꿔야만 Boot 코드가 동작한다
- c. 해당 보드에서의 부트로더의 주 역할은?
주로 바이너리를 다운로드, 업데이트를 할 때 사용된다
- d. 아두이노와 다르게 이 보드는 Bootloader가 매번 필요가 없는 이유는 ISP 장비로 ST-Link를 제공하고 있기 때문이다. 이는 해당 보드에 국한된 내용이 아닌 TI, 여타 보드(32F429ZI-DISC1도 해당)에도 적용된다. 그래서 ISP가 제공되는 보드는 적극적으로 이용할 수 있고, 없는 보드들은 자체 IAP를 개발해서 원하는 Flash에 어플리케이션을 다운로드해야 한다

1.2 Memory 관리

- 부트로더를 작성하기 위해서는 사용하고자 하는 MCU의 Memory map을 끄고 있어야만 한다. 따라서 이번에 예시 보드로 사용할 32F429ZI-DISC1의 Memory map을 한번 짚고 가려고 한다.

a. Memory map

제공하고 있는 docs 폴더에 stm32f429zi.pdf의 86페이지를 참고하도록 한다



b. Flash Memory(0x08000000, 2MB)

Flash Memory에는 우리가 작성한 main 프로그램이 들어있다. 어플리케이션이 동작하기 위해 필요한 코드와 데이터는 여기에 담겨 있다고 말할 수 있다. 비휘발성 메모리로 필요한 부분들은 프로그램 실행 시 RAM에 바로 적재된다. 다음은 Flash module의 bank 정보다

Table 6. Flash module - 2 Mbyte dual bank organization (STM32F42xxx and STM32F43xxx)

Block	Bank	Name	Block base addresses	Size	
Main memory	Bank 1	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes	
		Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes	
		Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes	
		Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes	
		Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes	
		Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes	
		Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes	
		-	-	-	
		-	-	-	
		-	-	-	
		Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes	
	Bank 2	Sector 12	0x0810 0000 - 0x0810 3FFF	16 Kbytes	
		Sector 13	0x0810 4000 - 0x0810 7FFF	16 Kbytes	
		Sector 14	0x0810 8000 - 0x0810 BFFF	16 Kbytes	
		Sector 15	0x0810 C000 - 0x0810 FFFF	16 Kbytes	
		Sector 16	0x0811 0000 - 0x0811 FFFF	64 Kbytes	
		Sector 17	0x0812 0000 - 0x0813 FFFF	128 Kbytes	
		Sector 18	0x0814 0000 - 0x0815 FFFF	128 Kbytes	
		-	-	-	
		-	-	-	
		-	-	-	
		Sector 23	0x081E 0000 - 0x081F FFFF	128 Kbytes	
System memory			0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes	
OTP			0x1FFF 7800 - 0x1FFF 7A0F	528 bytes	
Option bytes	Bank 1		0x1FFF C000 - 0x1FFF C00F	16 bytes	
	Bank 2		0x1FFE C000 - 0x1FFE C00F	16 bytes	

OTP는 One Time Programming의 약자로, 제품이 생산될 때 이미 한 번 프로그램이 된 메모리다. 여기에는 제품에 대한 설명들이 담겨 있다. 기본적으로 STM 프로그램은 Sector 0부터 writing이 쓰여진다

c. SRAM(0x20000000, 256K + 4K)

일반적인 RAM 메모리로 생각하면 되고, 읽기 쓰기가 가능한 메모리다. 주로 Stack, Heap 영역을 주 목적으로 사용하며, 휘발성 메모리다

d. System Memory(0x1FFF0000)

STM에서 제공하는 내장 부트로더가 저장되는 공간이다. 기본적으로는 이 영역에서 실행되지 않고 Boot Pin Configuration으로 설정해야만 바로 진입이 가능하다. 현재 STM 보드는 기본적으로 ISP가 내장되어 있기 때문에 사용하지는 않는다

1.3 Keil을 통한 Memory View

- 부트로더 프로그램은 Reset이 될 때 시작된다. MCU 프로그램에서 Reset이 될 때는 Reset Handler부터 시작이 된다. 그리고 그 전에 MCU의 MSP가 지정되어야만 RAM의 영역 시작점을 알게 된다. MCU가 리셋이 되면, 프로세서의 PC(Program Counter)는 0번지를 가리키게 된다. 보통 0번지에 위치한 값은 MSP 값이 들어있다. 그리고 4바이트 뒤에는 Reset Handler가 정의되어 있다.
- 그런데 ST의 Flash 시작은 0x08000000이다. 그런데 0번지부터 시작하면 앞뒤가 안맞게 된다. 사실 0번지 시작은 ARM에서 정의한 것이고 각 Vendor는 자신의 MCU에 따라서 다르게 지정할 수 있는 것이다. 따라서 STM은 0x08000000로 Aliasing을 시켰다고 볼 수 있다. 실제 TI는 ARM에서 정한 것과 같이 0x00000000에서 시작하기도 한다

Table 2-4. Memory Map

Start	End	Description	For details, see page ...
Memory			
0x0000.0000	0x0003.FFFF	On-chip Flash	540
0x0004.0000	0x1FFF.FFFF	Reserved	-
0x2000.0000	0x2000.7FFF	Bit-banded on-chip SRAM	525
0x2000.8000	0x21FF.FFFF	Reserved	-
0x2200.0000	0x220F.FFFF	Bit-band alias of bit-banded on-chip SRAM starting at 0x2000.0000	525
0x2210.0000	0x3FFF.FFFF	Reserved	-
Peripherals			
0x4000.0000	0x4000.0FFF	Watchdog timer 0	776
0x4000.1000	0x4000.1FFF	Watchdog timer 1	776
0x4000.2000	0x4000.3FFF	Reserved	-
0x4000.4000	0x4000.4FFF	GPIO Port A	658
0x4000.5000	0x4000.5FFF	GPIO Port B	658

A. Keil에서 확인 (Keil에 대해서 사용법을 모른다면 아래에서 Keil 설치 후에 따라할 것)

1.4 Boot Configuration

- STM에서는 자체 Technic 동작으로 메모리 Aliasing을 볼 수 있었다. 따라서 Boot Configuration이 정해지면, Reset Handler는 해당 메모리 번지로 aliasing되어 Jump하게 된다. 예를 들어 0x00000004에 접근하기 위해선 실제 물리주소인 0x08000004로 이동하게 되는 것이다. 어떻게 보면 벡터 테이블의 Offset이 0x08000000라고 할 수 있다

2.4 Boot configuration

Due to its fixed memory map, the code area starts from address 0x0000 0000 (accessed through the ICode/DCode buses) while the data area (SRAM) starts from address 0x2000 0000 (accessed through the system bus). The Cortex®-M4 with FPU CPU always fetches the reset vector on the ICode bus, which implies to have the boot space available only in the code area (typically, Flash memory). STM32F4xx microcontrollers implement a special mechanism to be able to boot from other memories (like the internal SRAM).

In the STM32F4xx, three different boot modes can be selected through the BOOT[1:0] pins as shown in [Table 2](#).

Table 2. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

기본적으로 Boot Configuration은 BOOT0은 0으로 Flash Memory 0번지로 Booting을 시작하게 된다. 이후 실습에서는 BOOT1과 0핀을 각각 0과 1로 두어서 내장 부트로더가 있는 System Memory로 Jump 해보려고 한다. 이전에 개발환경을 구성해보려고 한다

1.5 개발환경 구성

- 기존의 이클립스 기반의 IDE는 모든 OS에서 쉽게 사용할 수 있다는 장점이 있지만, 사용자가 시스템과 MCU를 이해하고 있어야만 컴파일러 등 환경설정과 확장이 쉽게 가능하다. 물론 TrueStudio나 OpenSTM을 사용하면 쉽게 구성이 하기도 하다. 그럼에도 불구하고 이번에는 Window에서만 사용이 가능하고 더 편리하다고 생각하는 Keil-MDK5를 사용하도록 할 것이다

A. Keil 사이트(<https://www2.keil.com/mdk5>)에 접속하도록 한다

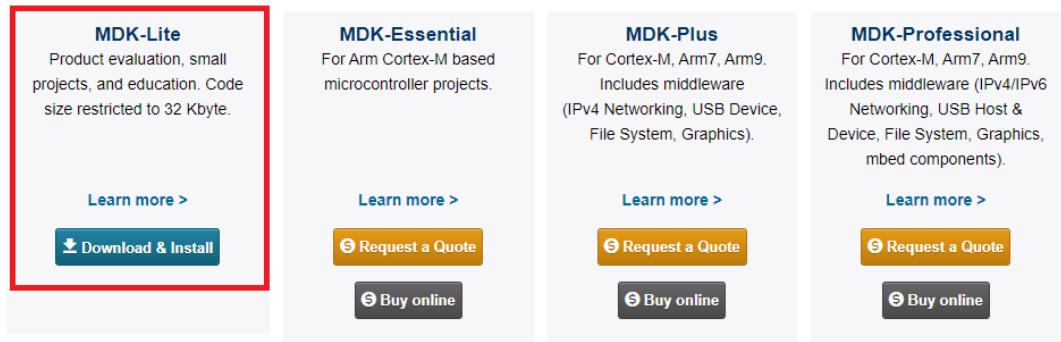
The screenshot shows the Keil MDK homepage. At the top, there's a navigation bar with links for Home, Products, Download, Events, Support, and a search bar. Below the navigation is a banner for 'MDK Microcontroller Development Kit'. The banner includes a download link for 'MDK v5.25' and a note about 'Update: MDK v5.29 contains Arm Compiler 6.13 and enhancements in μVision for debugging and project build.' To the right of the banner are two columns of 'Quick Links' and 'Software Packs'. The 'Quick Links' column lists 'Getting Started', 'Online Manuals', 'Compare Editions', 'Middleware', and 'Learn'. The 'Software Packs' column lists 'Device List', 'Evaluation Boards', 'MDK v4 Legacy Support', 'Third-Party Software Packs', and 'Public Software Packs'. Below the banner, there's a section titled 'Product Components' with two main categories: 'MDK Tools' and 'Software Packs'. Under 'MDK Tools', there are boxes for 'MDK-Core' (μVision IDE & debugger with pack management) and 'Arm C/C++ Compiler' (With safety qualification). Under 'Software Packs', there are boxes for 'Device' (Startup, CMSIS, Device HAL, CMSIS Drivers), 'CMSIS' (CMSIS-Core, CMSIS-DSP, CMSIS-RTOS), 'MDK-Middleware' (Network (IPv4/IPv6), USB Host/Device, File System, Graphics, Mbed TLS, SSL/TLS security, IoT connectors), and 'MDK' (based on μVision (Windows only) with leading support for Cortex-M devices including the new Armv8-M architecture). At the bottom of the page, there are several small paragraphs of text providing more details about the software components.

사이트 설명을 보면 IDE는 컴파일러, 여러 Device, ARM CMSIS, Middleware를 모두 지원하는 것으로 보인다

- B. 현재 창에서 더 밑으로 내려가면 MDK Edition 목록으로 나열되어 있다. 32K 코드 사이즈까지 지원하는 무료 버전을 다운로드 받도록 한다. 삼용 프로그램이지만, 현재 작성하는 부트로더 프로그램은 가볍기 때문에 충분한 환경으로 여겨진다

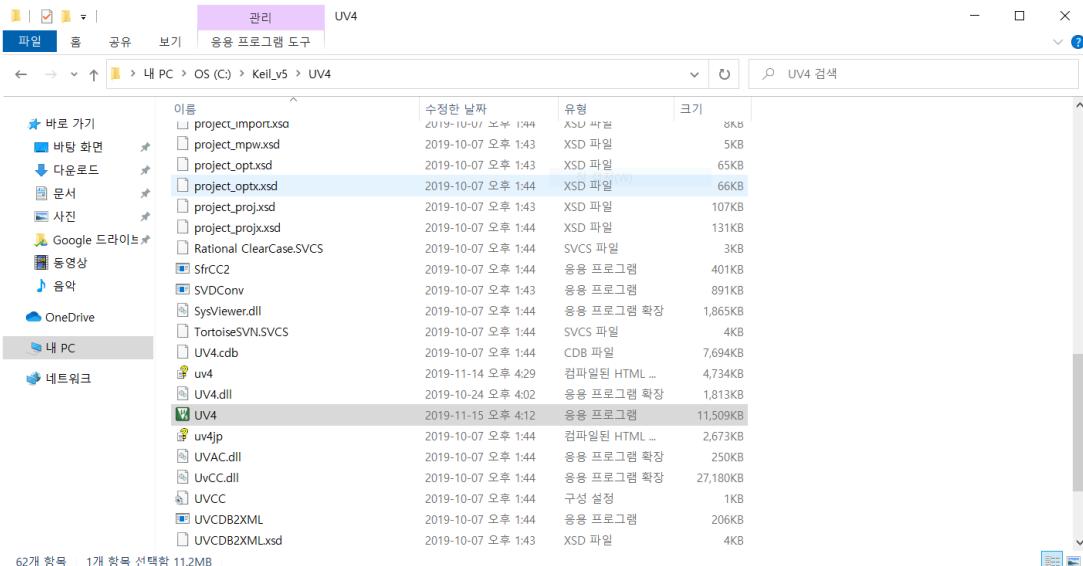
MDK Editions

MDK is available in various editions. [Compare Editions >](#)

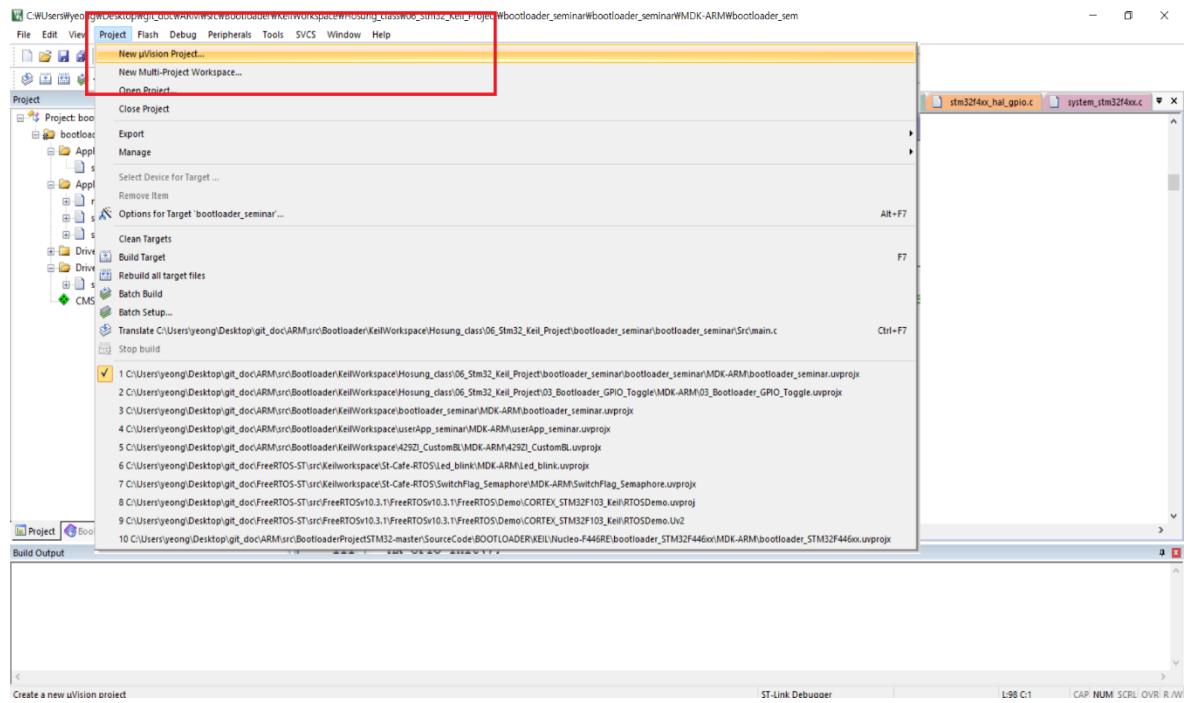


MDK is also available as part of [Arm Development Studio >>](#)

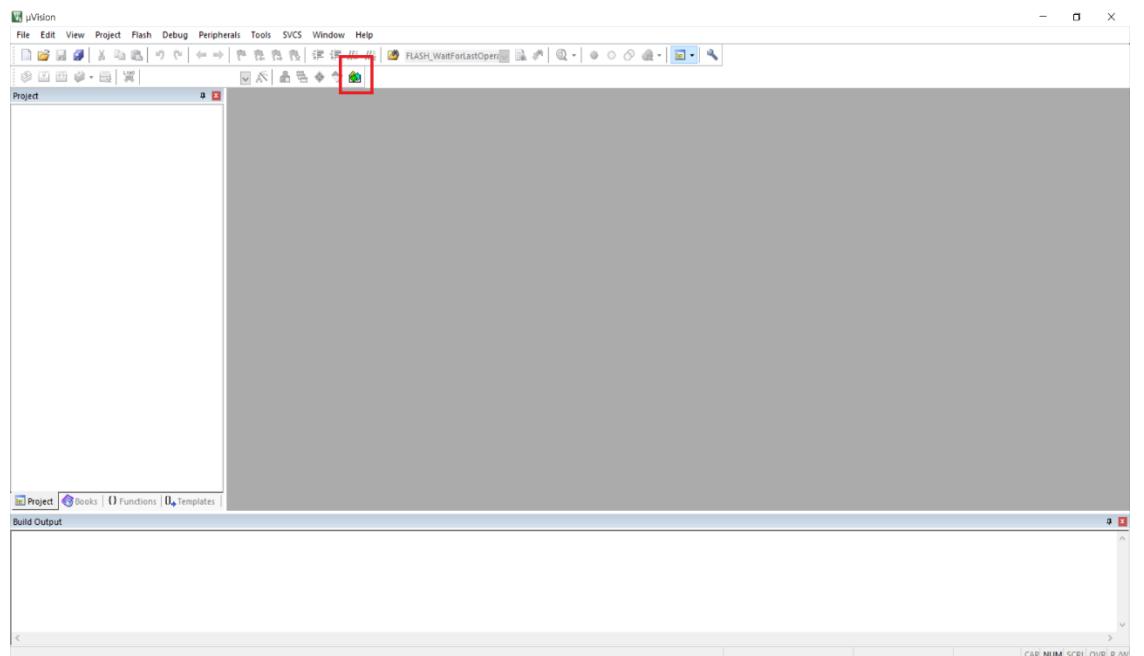
- C. 다운로드 받은 설치파일을 기본 설정으로 두고 다운로드를 시작한다. 설치가 모두 완료되면 기본 경로로 설치했으니, 다음 경로(C:/Keil_v5/UV4)로 이동하면 UV4 IDE를 찾을 수 있다. 따라서 이를 실행하도록 한다



D. 적절한 경로에 폴더를 생성하고, 프로젝트를 생성할 것이다. 프로젝트 생성은 Project 탭에서 할 수 있다

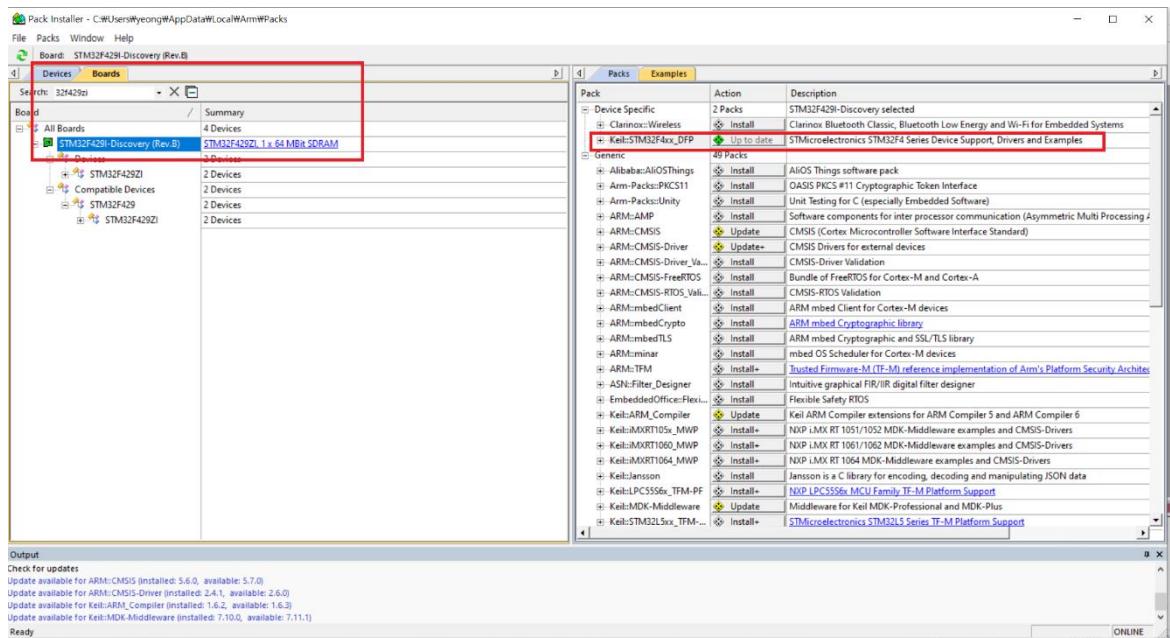


E. 하지만 프로젝트를 생성하기 위해서 Target을 잡아야 하는데, Target 파일이 모두 설치되지 않고, 원하는 Target 파일을 사용자가 직접 선정해야만 한다. 따라서 Pack Installer를 실행하도록 한다

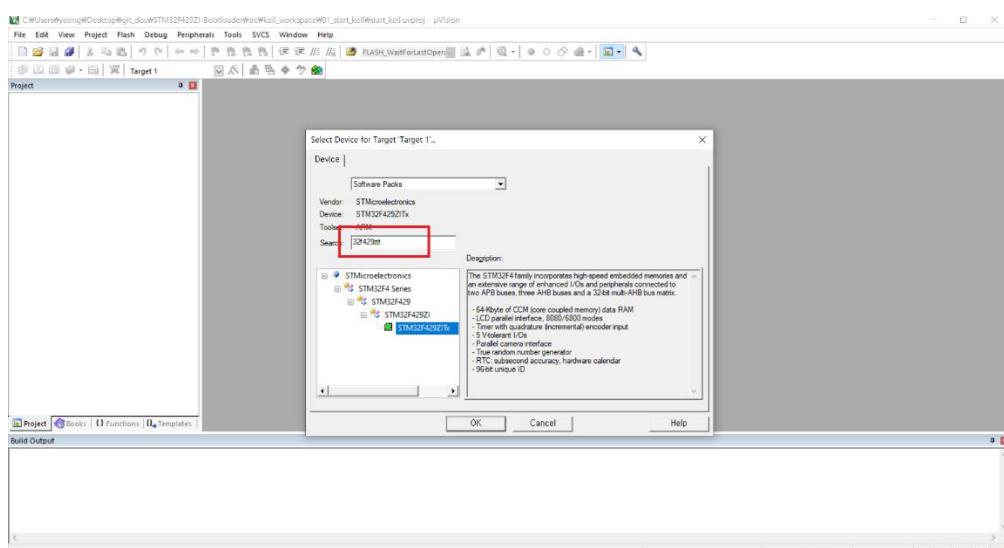


- F. Board 탭을 클릭해서 '32f429zi'를 작성하면 설치 목록이 나오게 된다. MCU 개발에 있어 기본적으로 필요한 드라이버와 startup과 같은 파일은 DFP에 담겨 있다. 그리고 보드 자체로 다운로드를 받으면 간단한 예제파일 몇개도 다운로드를 해주게 된다. 처음에는 Online 경로를 통해 패키지 목록을 받기 때문에 시간이 걸릴 수도 있다. 아래 Online에 퍼센트가 증가하는 것을 볼 수 있을 것이다

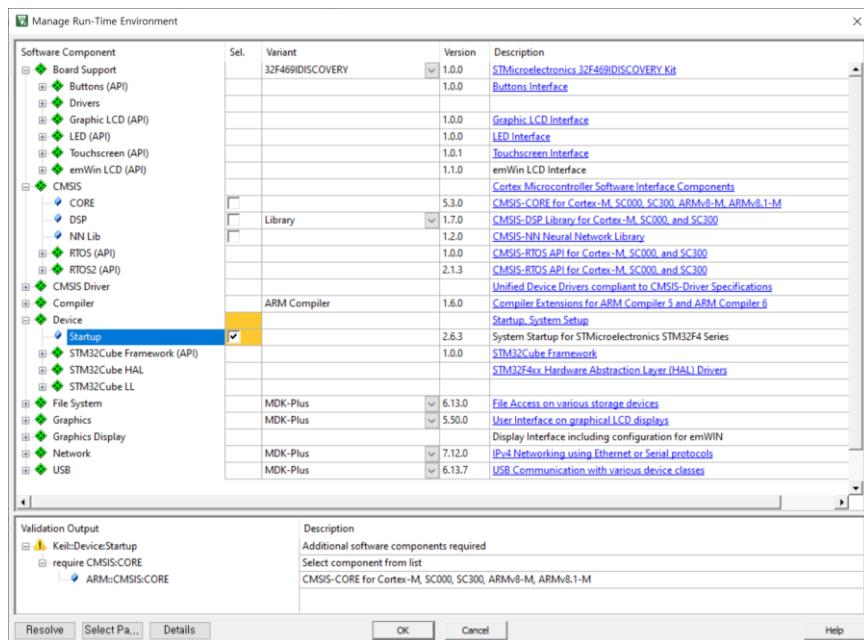
설치를 마치면 해당 패키지가 up to date로 될 것이다.



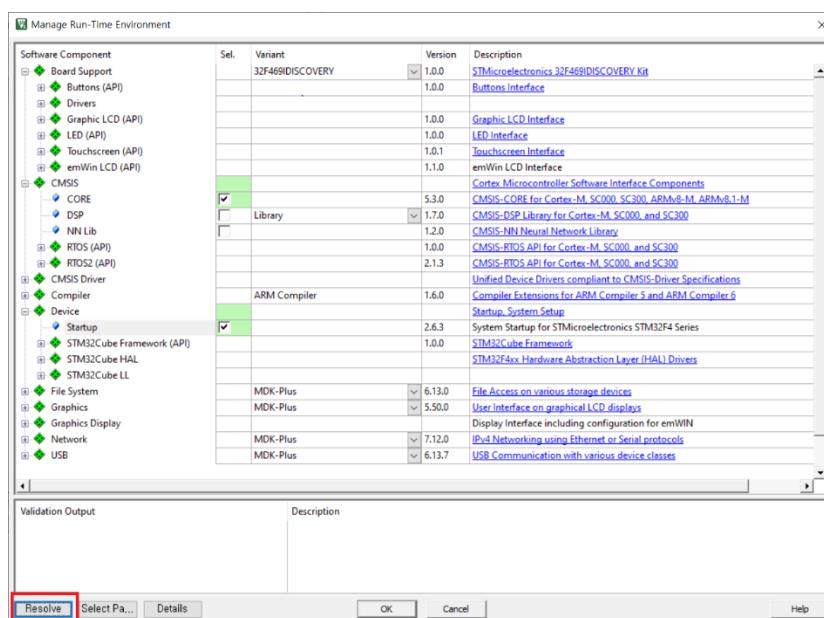
- G. 설치가 완료되었다면 pack installer 창을 끄도록 한다. 그리고 아까 하고자 했던 프로젝트 생성 버튼을 클릭하도록 한다. 그리고 search 창에 '32f429xit'을 입력하도록 한다



H. 그러면 Manage Run-Time Environment 창이 뜨게 된다. 우리는 이 창을 통해서 해당 보드의 Led blink 예제 프로그램을 이용해보려고 한다. 이 창에서 보통 먼저 하는 작업이면서 가장 중요한 작업은 Startup 파일을 생성하는 것이다. Device 탭에서 클릭함으로써 생성할 수 있다

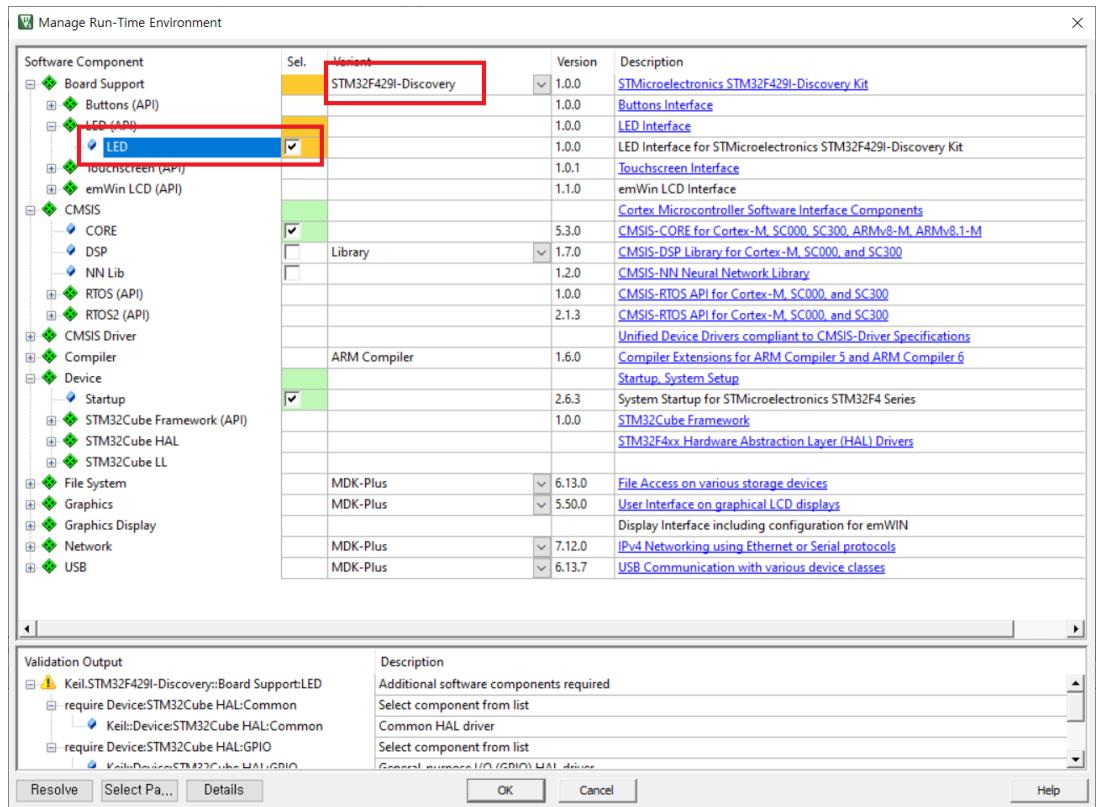


I. 이 창에서 노란색이 뜬 것은 dependency 관련한 주의사항이다. 즉 해당 설정을 100% 완료하기 위해선 dependency를 해결해야만 한다. 이는 왼쪽 하단에 'resolve'를 눌러 쉽게 해결할 수 있다

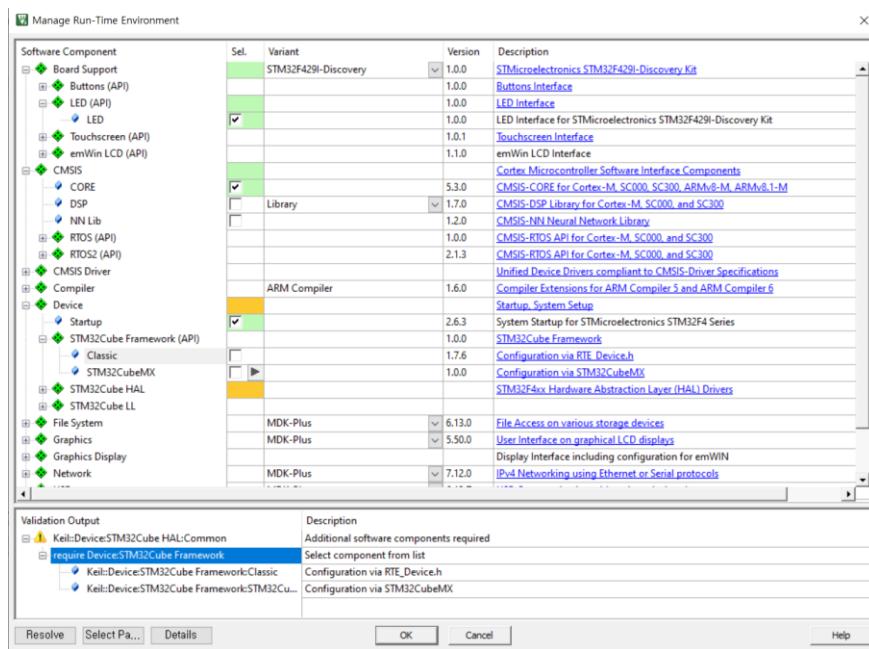


누르는 순간 CORE쪽이 체크가 되면서 초록색으로 바뀌는 것을 볼 수 있다

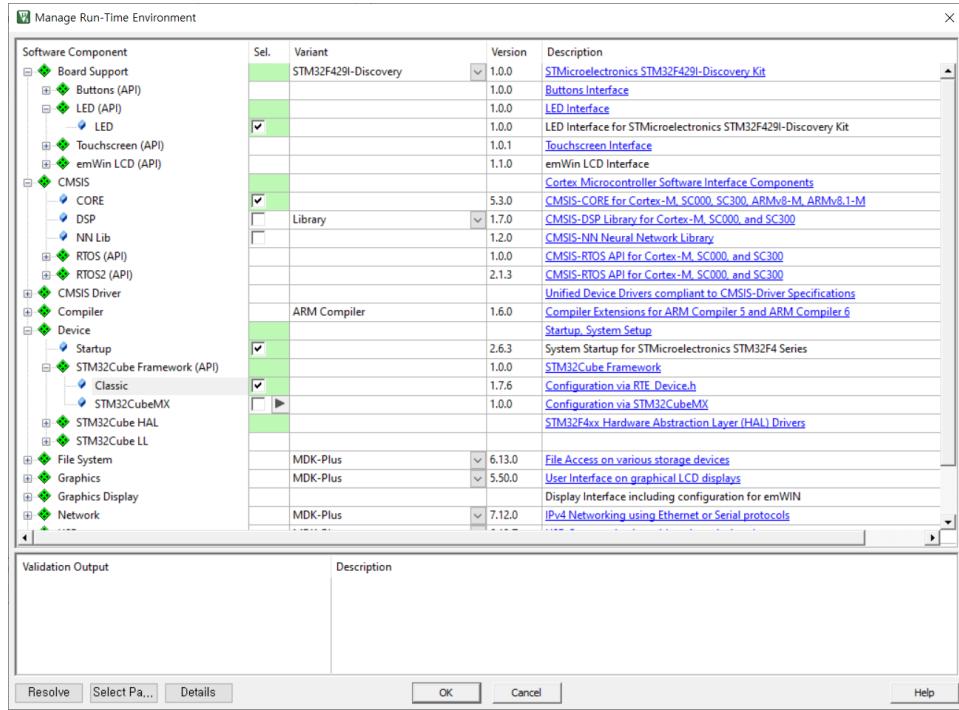
- J. 그리고 BSP로 주어지는 Led blink를 활용할 것이다. 맨 위에 Board Support 탭에서 반드시 보드를 올바르게 맞춰준 후 LED 부분을 클릭하도록 한다



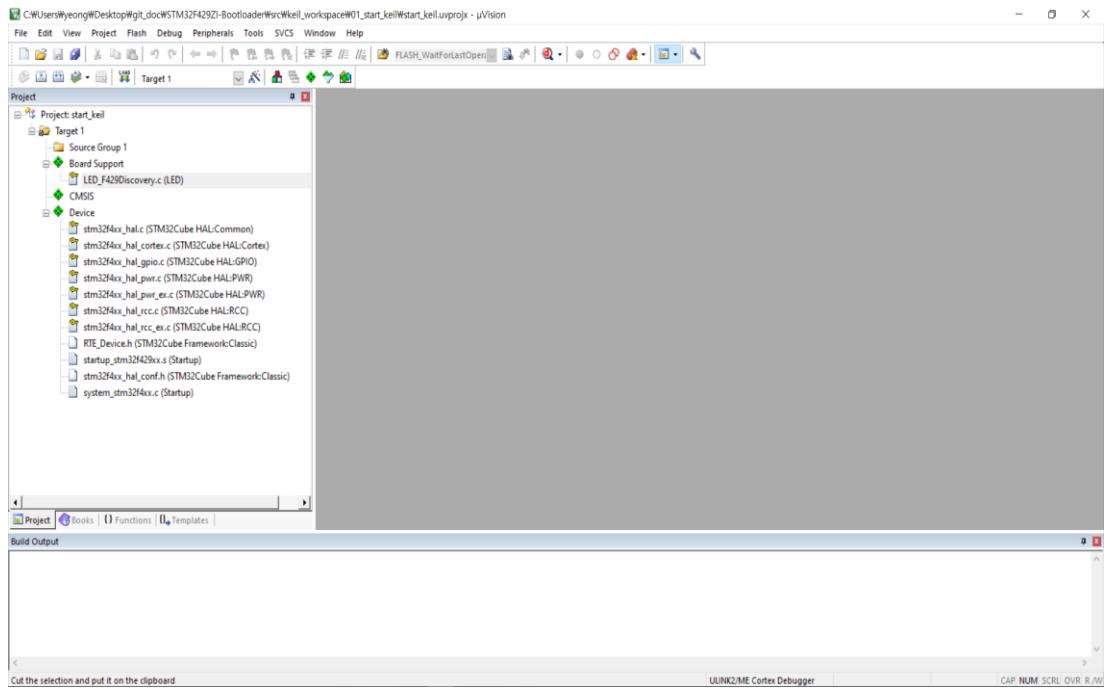
그러면 다시 dependency 주의가 나오게 된다. Resolve를 통해 해결해보도록 한다.



그래도 아직 해결되지 않는다. 이는 사용자에게 라이브러리 설정에 대한 framework를 설정하라는 것이다. 현재는 따로 cubemx라는 Tool을 이용하지 않을 것이기 때문에, Classic으로 설정할 것이다



그리고 확인을 누르면 프로젝트가 생성된 것을 확인할 수 있다



1.6 Keil 프로젝트 분석

- 위에서의 마지막 그림과 같이 프로젝트가 여러 Tree 형태로 구성된 것을 알 수 있다. 중요한 파일 몇 가지를 간략하게 살펴보려고 한다. 이 파일들은 추후에 부트로더 개발을 함께 있어 중요한 역할들을 하게 된다

A. startup_stm32f429xx.s

- startup 파일로, 전원을 켰을 시에 main 시작 전에 동작하는 코드가 된다

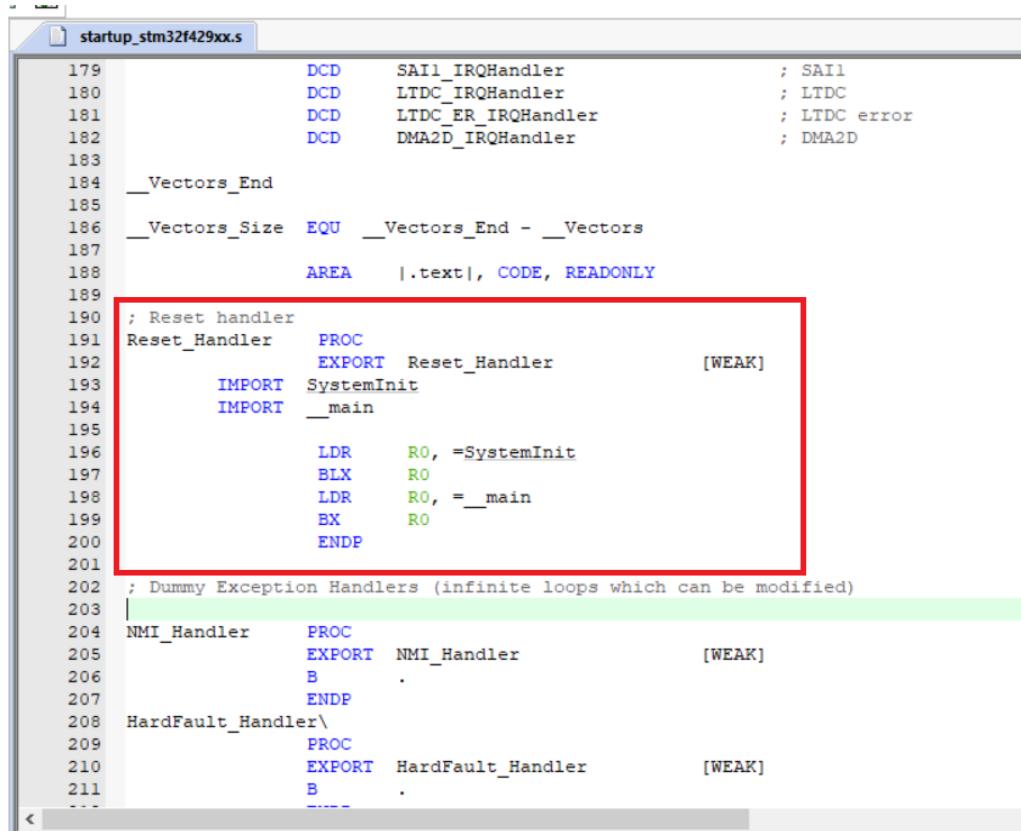
a. stack, heap 사이즈를 설정한다

```
43 ; <o> Stack Size (in Bytes) <0x0-0xFFFFFFFF:8>
44 ; </h>
45
46 Stack_Size      EQU      0x00000400
47
48             AREA     STACK, NOINIT, READWRITE, ALIGN=3
49 Stack_Mem       SPACE    Stack_Size
50 __initial_sp
51
52
53 ; <h> Heap Configuration
54 ; <o>  Heap Size (in Bytes) <0x0-0xFFFFFFFF:8>
55 ; </h>
56
57 Heap_Size        EQU      0x00000200
58
59             AREA     HEAP, NOINIT, READWRITE, ALIGN=3
60 __heap_base
61 Heap_Mem         SPACE    Heap_Size
62 __heap_limit
63
64             PRESERVE8
65             THUMB
66
67
68 ; Vector Table Mapped to Address 0 at Reset
69             AREA     RESET, DATA, READONLY
70             EXPORT   __Vectors
71             EXPORT   __Vectors_End
72             EXPORT   __Vectors_Size
73
74 __Vectors        DCD      __initial_sp           ; Top of Stack
75             DCD      Reset_Handler          ; Reset Handler
76             DCD      NMI_Handler           ; NMI Handler
77             DCD      HardFault_Handler      ; Hard Fault Handler
78             DCD      MemManage_Handler      ; MPU Fault Handler
79             DCD      BusFault_Handler       ; Bus Fault Handler
80             DCD      UsageFault_Handler     ; Usage Fault Handler
81             DCD      0                   ; Reserved
82             DCD      0                   ; Reserved
83             DCD      0                   ; Reserved
84             DCD      0                   ; Reserved
85             DCD      SVC_Handler          ; SVCall Handler
86             DCD      DebugMon_Handler      ; Debug Monitor Handler
87             DCD      0                   ; Reserved
88             DCD      PendSV_Handler       ; PendSV Handler
89             DCD      SysTick_Handler       ; SysTick Handler
90
91             ; External Interrupts
92             DCD      WWDG_IRQHandler      ; Window WatchDog
93             DCD      PVD_IRQHandler        ; PVD through EXTI Line detection
94             DCD      TAMPER_STAMP_IRQHandler ; Tamper and TimeStamps through the EXTI line
95             DCD      RTC_WKUP_IRQHandler    ; RTC Wakeup through the EXTI line
96             DCD      FLASH_IRQHandler      ; FLASH
97             DCD      RCC_IRQHandler        ; RCC
98             DCD      EXTI0_IRQHandler      ; EXTI Line0
99             DCD      EXTI1_IRQHandler      ; EXTI Line1
100            DCD      EXTI2_IRQHandler      ; EXTI Line2
```

b. vector table

```
68 ; Vector Table Mapped to Address 0 at Reset
69             AREA     RESET, DATA, READONLY
70             EXPORT   __Vectors
71             EXPORT   __Vectors_End
72             EXPORT   __Vectors_Size
73
74 __Vectors        DCD      __initial_sp           ; Top of Stack
75             DCD      Reset_Handler          ; Reset Handler
76             DCD      NMI_Handler           ; NMI Handler
77             DCD      HardFault_Handler      ; Hard Fault Handler
78             DCD      MemManage_Handler      ; MPU Fault Handler
79             DCD      BusFault_Handler       ; Bus Fault Handler
80             DCD      UsageFault_Handler     ; Usage Fault Handler
81             DCD      0                   ; Reserved
82             DCD      0                   ; Reserved
83             DCD      0                   ; Reserved
84             DCD      0                   ; Reserved
85             DCD      SVC_Handler          ; SVCall Handler
86             DCD      DebugMon_Handler      ; Debug Monitor Handler
87             DCD      0                   ; Reserved
88             DCD      PendSV_Handler       ; PendSV Handler
89             DCD      SysTick_Handler       ; SysTick Handler
90
91             ; External Interrupts
92             DCD      WWDG_IRQHandler      ; Window WatchDog
93             DCD      PVD_IRQHandler        ; PVD through EXTI Line detection
94             DCD      TAMPER_STAMP_IRQHandler ; Tamper and TimeStamps through the EXTI line
95             DCD      RTC_WKUP_IRQHandler    ; RTC Wakeup through the EXTI line
96             DCD      FLASH_IRQHandler      ; FLASH
97             DCD      RCC_IRQHandler        ; RCC
98             DCD      EXTI0_IRQHandler      ; EXTI Line0
99             DCD      EXTI1_IRQHandler      ; EXTI Line1
100            DCD      EXTI2_IRQHandler      ; EXTI Line2
```

그리고 reset handler에 대한 정의를 확인할 수 있다. SystemInit 함수로 Clock setting과 같은 과정을 거쳐 main으로 Jump하게 된다

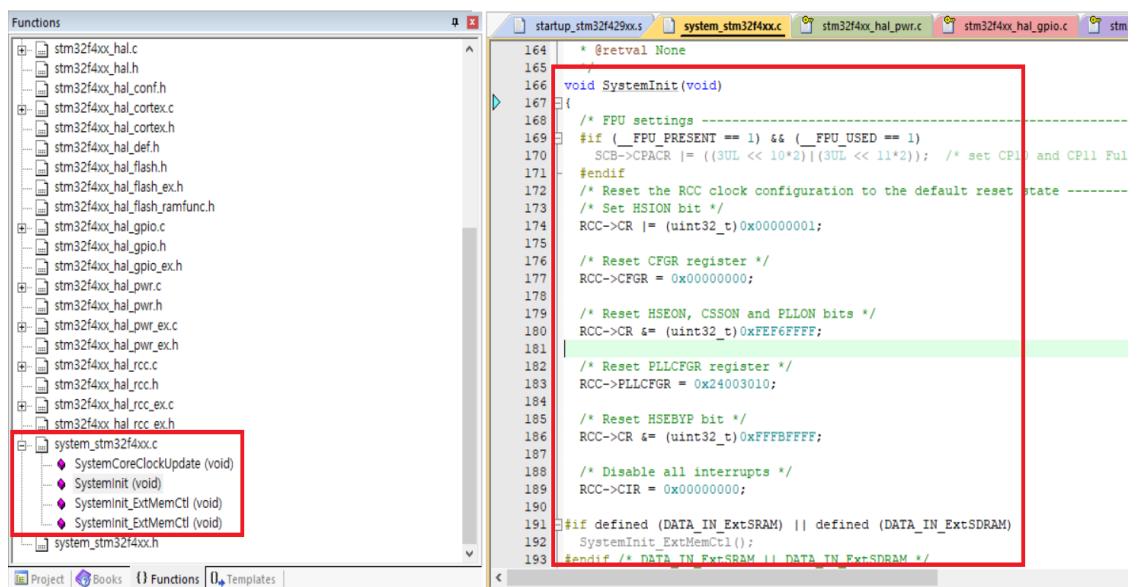


```

179          DCD    SAI1_IRQHandler           ; SAI1
180          DCD    LTDC_IRQHandler        ; LTDC
181          DCD    LTDC_ER_IRQHandler     ; LTDC error
182          DCD    DMA2D_IRQHandler       ; DMA2D
183
184      _Vectors_End
185
186      _Vectors_Size EQU _Vectors_End - _Vectors
187
188      AREA    .text!, CODE, READONLY
189
190      ; Reset handler
191      Reset_Handler    PROC
192          EXPORT  Reset_Handler        [WEAK]
193          IMPORT  SystemInit
194          IMPORT  __main
195
196          LDR    R0, =SystemInit
197          BLX    R0
198          LDR    R0, =__main
199          BX     R0
200          ENDP
201
202      ; Dummy Exception Handlers (infinite loops which can be modified)
203
204      NMI_Handler    PROC
205          EXPORT  NMI_Handler         [WEAK]
206          B     .
207          ENDP
208      HardFault_Handler\
209          PROC
210          EXPORT  HardFault_Handler   [WEAK]
211          B     .
212

```

SystemInit 함수의 정의는 system_stm32f4xx.c 파일에서 찾을 수 있다. **함수 내용은 rcc clock setting과 vector table의 offset을 설정하게 된다**



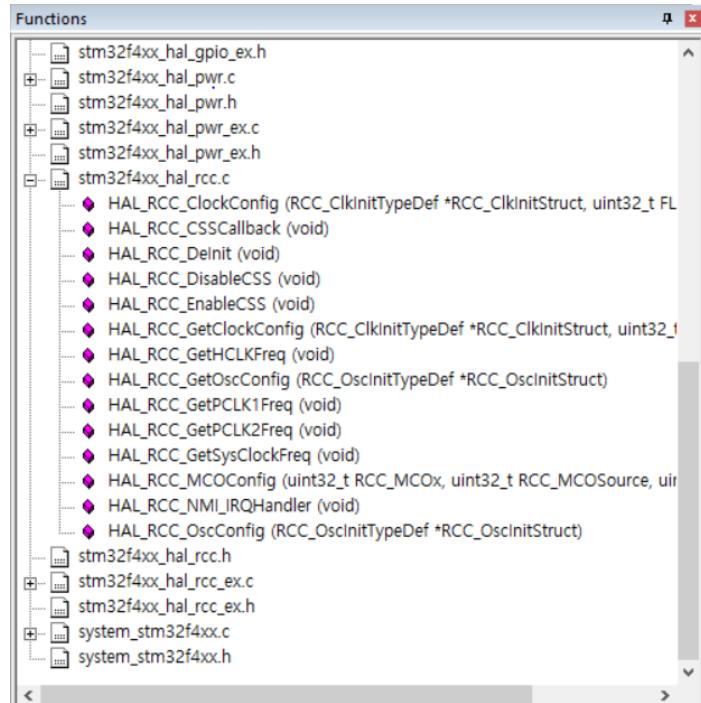
```

164  * @retval None
165
166 void SystemInit(void)
167 {
168
169     /* FPU settings -----
170     #if (_FPU_PRESENT == 1) && (_FPU_USED == 1)
171     SCB->CPACR |= ((SUL << 10*2)|(SUL << 11*2)); /* set CP1I and CP1F Full
172     #endif
173
174     /* Reset the RCC clock configuration to the default reset state -----
175     RCC->CR |= (uint32_t)0x00000001;
176
177     /* Reset CFGR register */
178     RCC->CFGR = 0x00000000;
179
180     /* Reset HSION, CSSON and PLLION bits */
181     RCC->CR &= (uint32_t)0xFEF6FFFF;
182
183     /* Reset PLLCFGR register */
184     RCC->PLLCFGR = 0x24003010;
185
186     /* Reset HSEBYP bit */
187     RCC->CR &= (uint32_t)0xFFFFBFFFF;
188
189     /* Disable all interrupts */
190     RCC->CIR = 0x00000000;
191
192     #if defined (DATA_IN_ExtSRAM) || defined (DATA_IN_ExtSDRAM)
193         SystemInit_ExtMemCtl();
194     #endif /* DATA_IN_ExtSRAM || DATA_IN_ExtSDRAM */

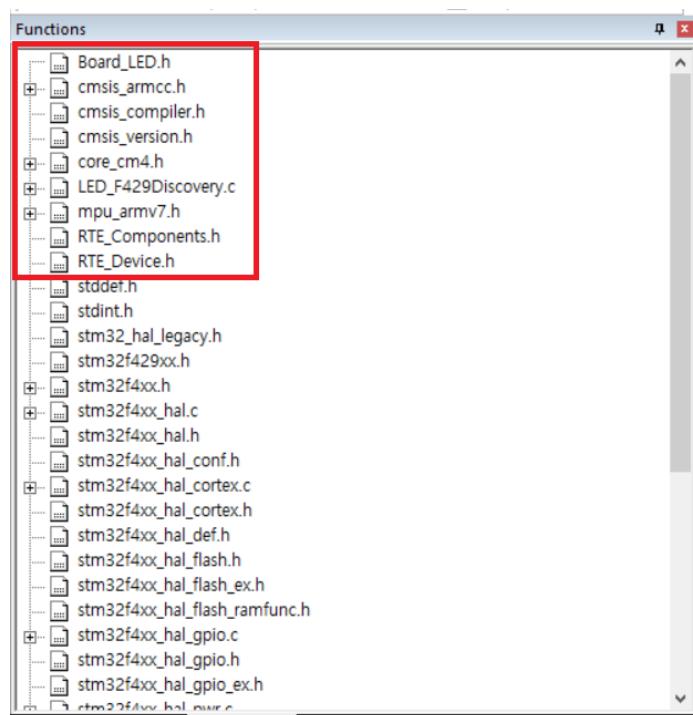
```

B. Peripheral Driver

- Keil에서는 hal 라이브러리로 주변장치를 제어하게 된다. 아래와 같은 라이브러리 함수로 쉽게 초기화가 가능하다



C. ARM CMSIS, BSP 관련 파일들

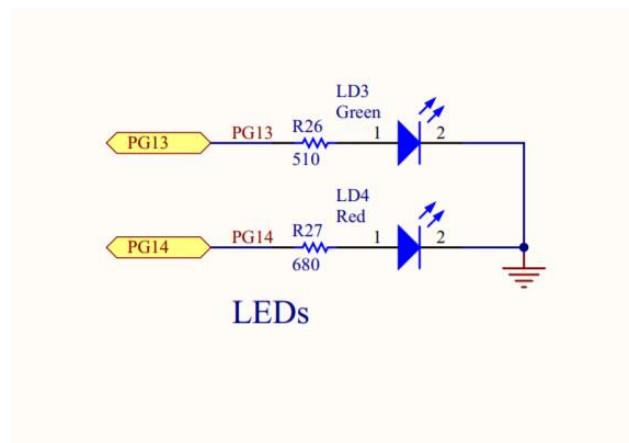


1.7 Led blink 파일 작성

- 본격적으로 BSP를 활용한 Application 프로그램을 작성하면서 Keil 환경 구성을 점검해보려고 한다

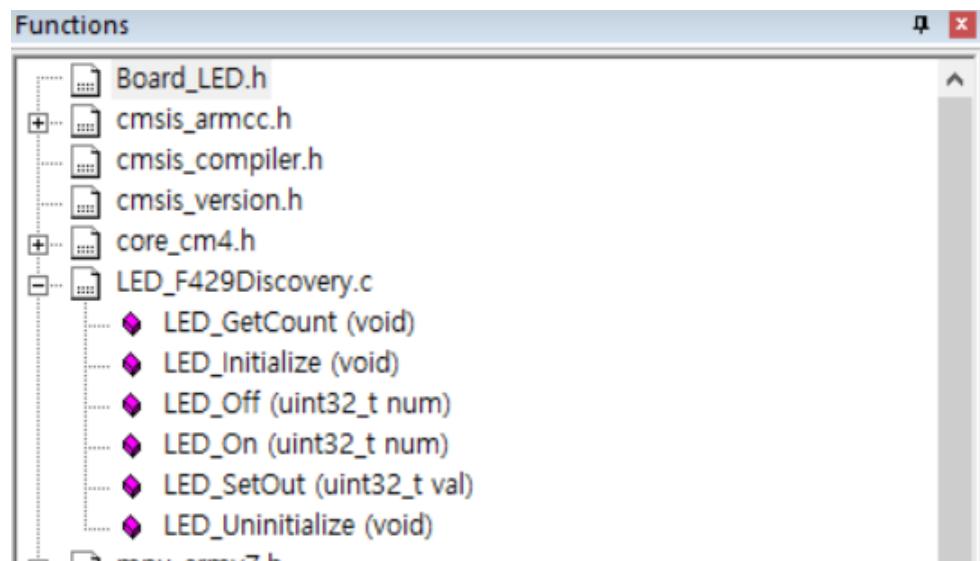
A. 회로도 확인

- PG13, PG14가 LED가 위치해 있다. 1을 입력하면 켜지는 회로다



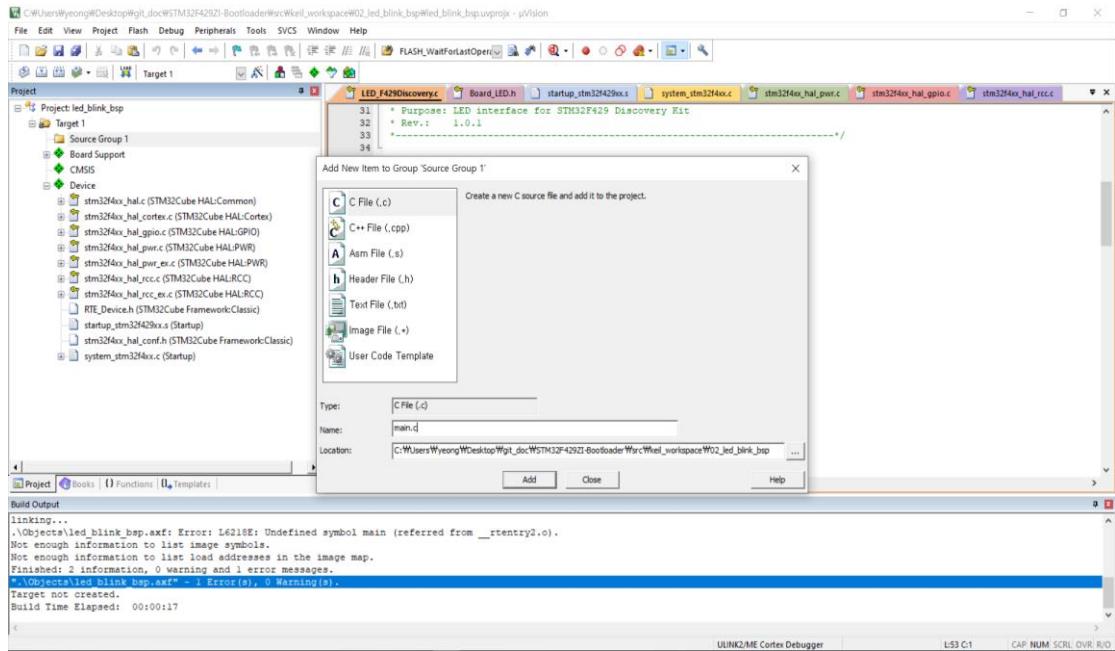
B. BSP 소스파일 확인

- 다음과 같은 라이브러리 함수 이용할 수 있다. 함수 이름이 직관적이라 사용이 편하다. 그리고 이를 이용하기 위해서는 'Board_LED.h' 파일을 포함시켜야 한다



C. main.c 소스파일 추가

- Source Group에 오른쪽 버튼을 눌러 'Add New Item to Group ...'을 눌러 C 소스파일을 추가할 수 있다



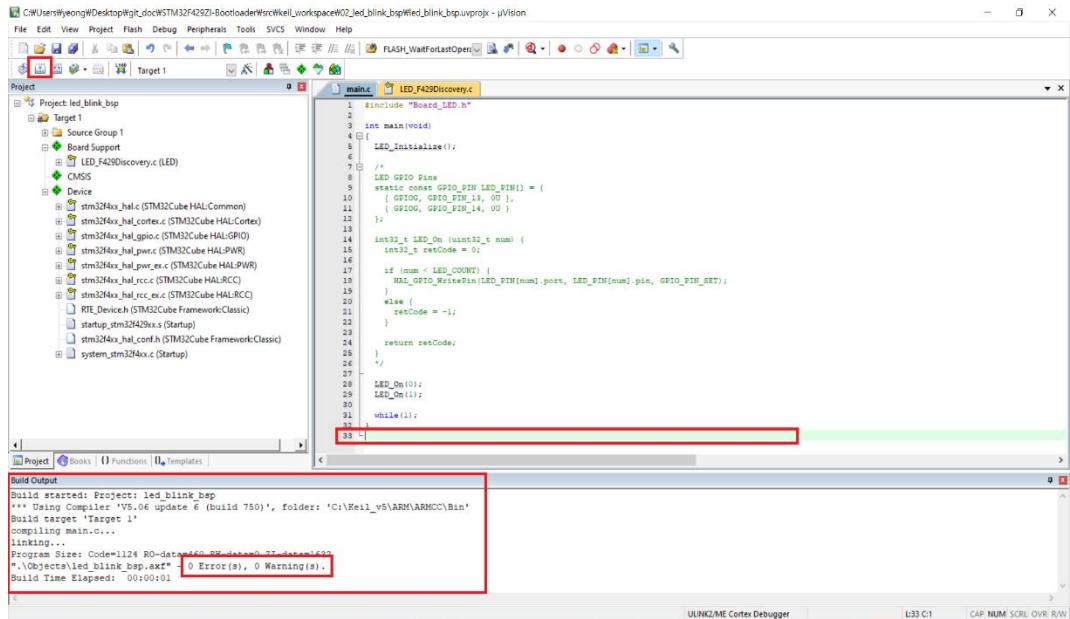
그리고 다음과 같이 소스파일을 작성할 수 있다. 소스파일에 대한 내용은 스스로 분석하는 것을 요한다.

```

main.c          LED_F429Discovery.c
1  #include "Board_LED.h"
2
3  int main(void)
4  {
5      LED_Initialize();
6
7      /*
8       * LED GPIO Pins
9       * static const GPIO_PIN LED_PIN[] = {
10       *     { GPIOG, GPIO_PIN_13, OU },
11       *     { GPIOG, GPIO_PIN_14, OU }
12     };
13
14     int32_t LED_On (uint32_t num) {
15         int32_t retCode = 0;
16
17         if (num < LED_COUNT) {
18             HAL_GPIO_WritePin(LED_PIN[num].port, LED_PIN[num].pin, GPIO_PIN_SET);
19         }
20         else {
21             retCode = -1;
22         }
23
24         return retCode;
25     }
26
27
28     LED_On(0);
29     LED_On(1);
30
31     while(1);
32 }

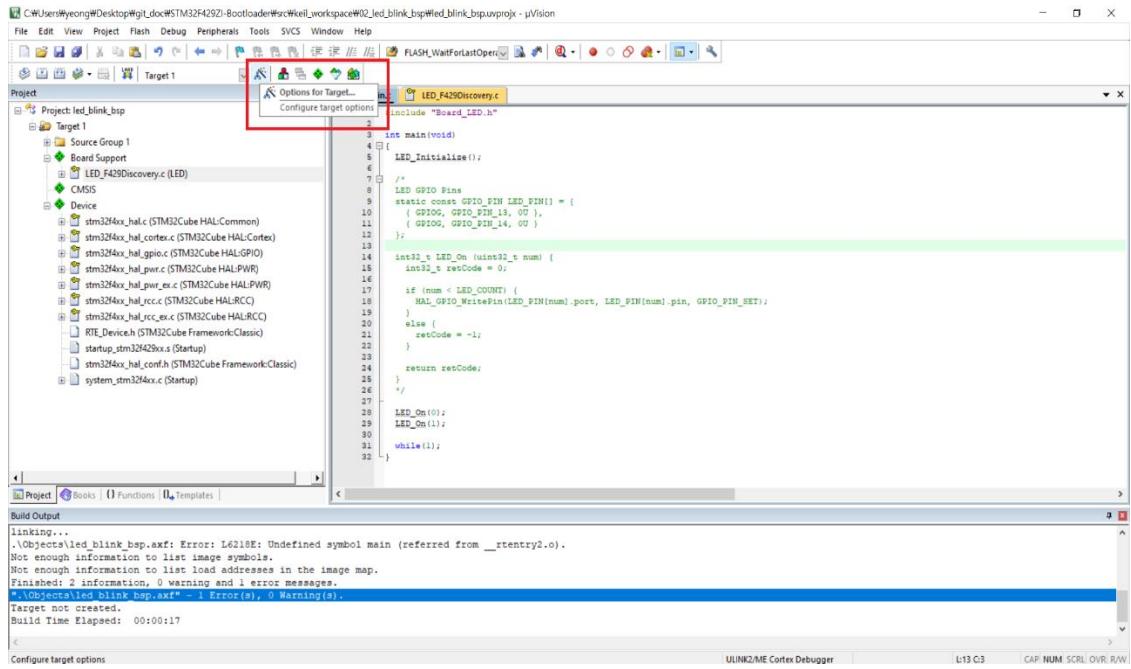
```

그리고 빌드를 하면 오류가 없어야 정상이다. 그리고 warning으로 발생하는 것은 Keil에서 마지막 줄은 개행이 아니기 때문이다. 이유는 모르겠지만 Keil에서 마지막 줄은 비워야만 warning이 사라진다

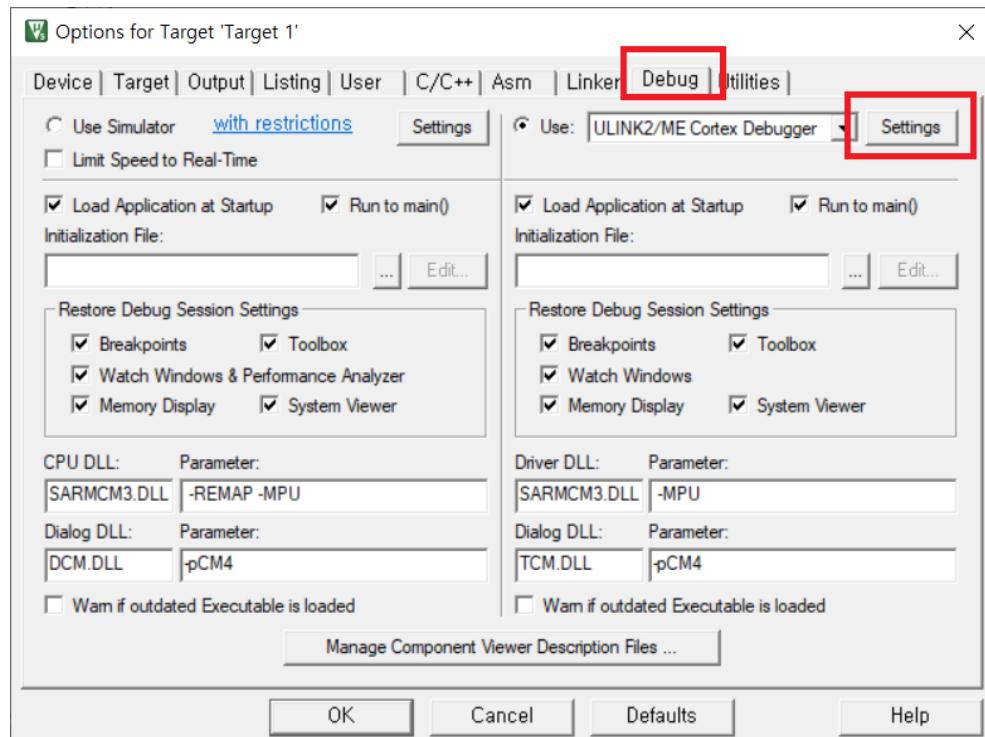


D. Target Flash Option

- Target Flash를 위한 옵션을 추가적으로 Debugger 설정을 해야 한다. 여러 Debugger 장치가 있기 때문에, 사용자가 직접 설정해야만 한다. 기본적인 Setting은 이미 Target 설정에서 Keil이 자동적으로 맞춰주게 된다. 'Option for Target'을 누르도록 한다

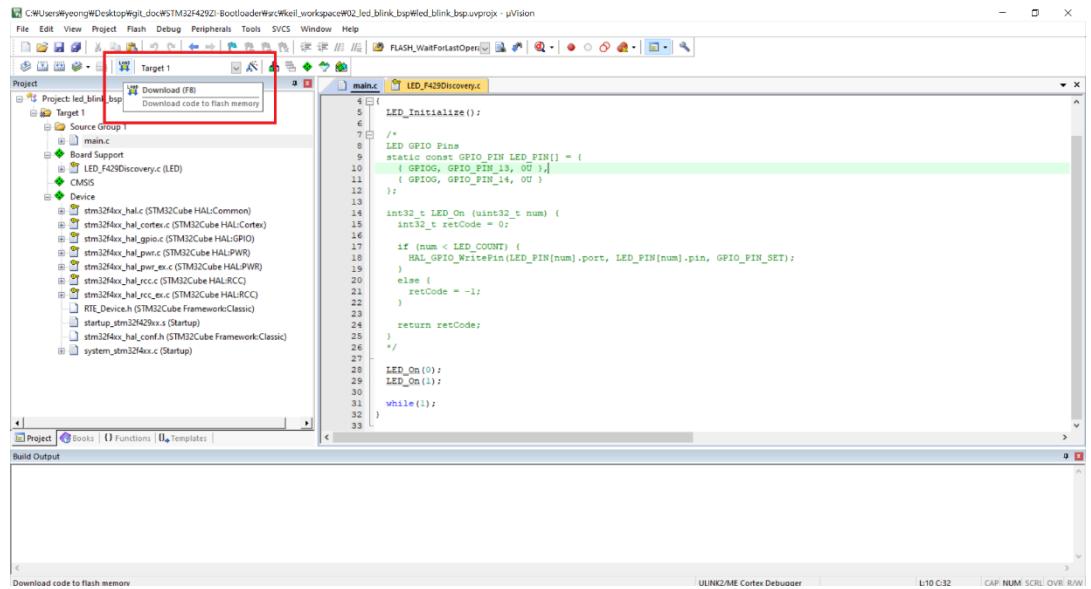


- a. Debug 탭에서 Setting을 누른다. ST-Link Debugger로 설정할 것이다

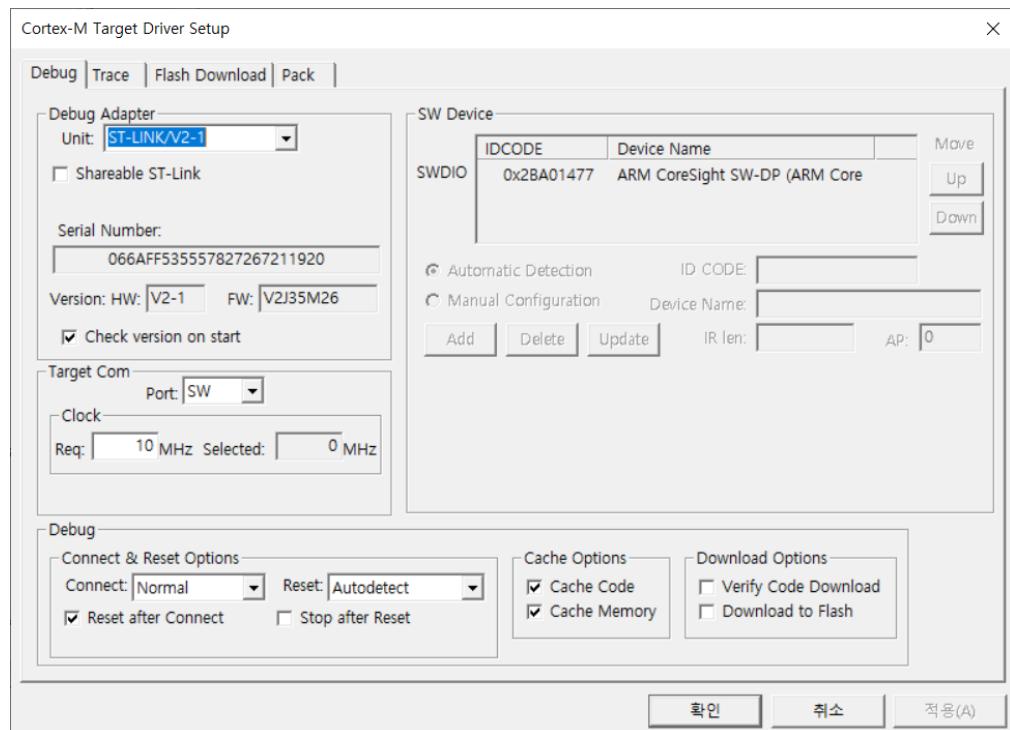


- b. ST 보드를 잘 연결한 후에 Settings를 누르고 들어가면 자동적으로 장치를 인식해서 정보를 출력할 것이다. 여기서 중요한 옵션이 reset and run을 체크하는 것이다. 다운로드 후 바로 다운로드한 프로그램을 바로 실행하는 것이다

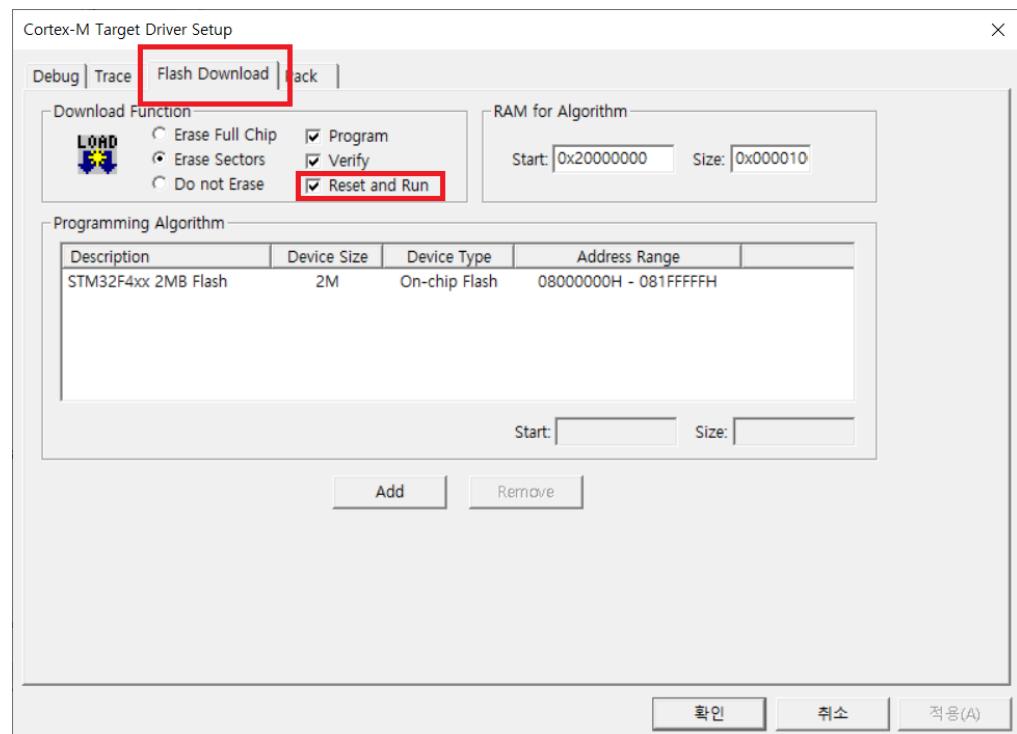
c. 설정을 완료했다면 다운로드를 하도록 한다



d. 보드 상태를 확인하도록 한다. 자동으로 제품 정보가 출력되는 것을 볼 수 있다



e. 그리고 reset and run 옵션을 추가하도록 한다



f. 그리고 보드에 다운로드를 해서 보드 상태를 확인하도록 한다

<사진>

1.8 OpenSTM32 System Workbench 개발환경 구축

- 지금까지 작업한 Keil은 원도우에서만 지원되는 IDE다. 따라서 다른 운영체제라면 다른 IDE를 고려해야만 한다. 그 중 하나의 대안이 OpenSTM32 System-Workbench다. 무료이면서 코드 사이즈도 제한이 없을뿐더러 Eclipse 기반으로 OS 제약도 없다는 것이 큰 장점이다. 설치를 위해 <https://www.openstm32.org/HomePage> 링크로 접속하도록 한다. 그리고 계정을 생성하고 로그인을 한다. 다음 원하는 OS 용을 다운로드 받도록 한다

A. 다운로드

링크

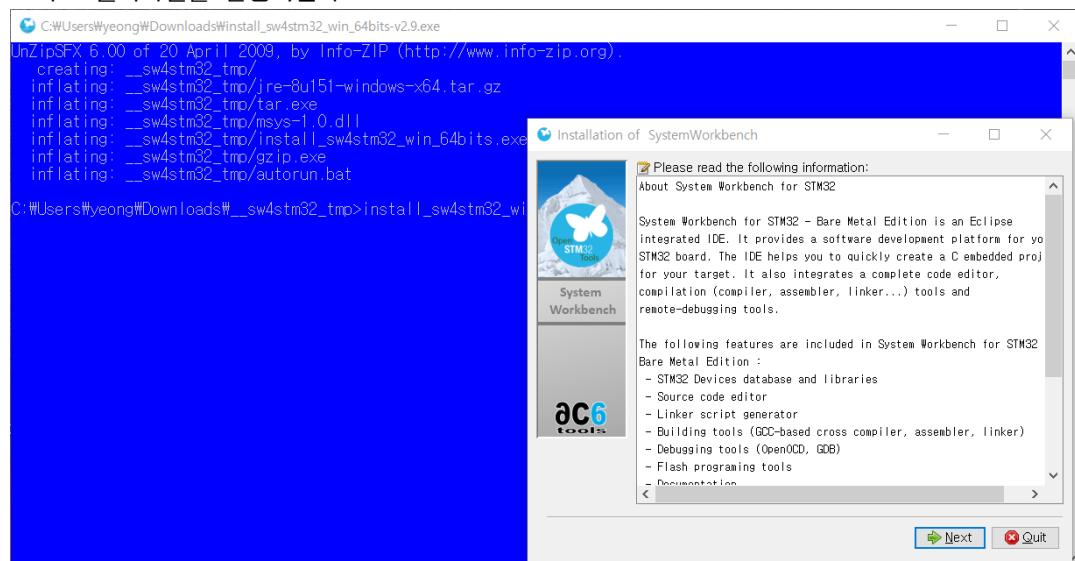
(<https://www.openstm32.org/Downloading%2Bthe%2BSYSTEM%2BWorkbench%2Bfor%2BSTM32%2Binstaller>)로 들어가도록 한다. 그리고 현재 OS는 원도우이므로 원도우용으로 다운로드 할 것이다

Windows 7 & 10

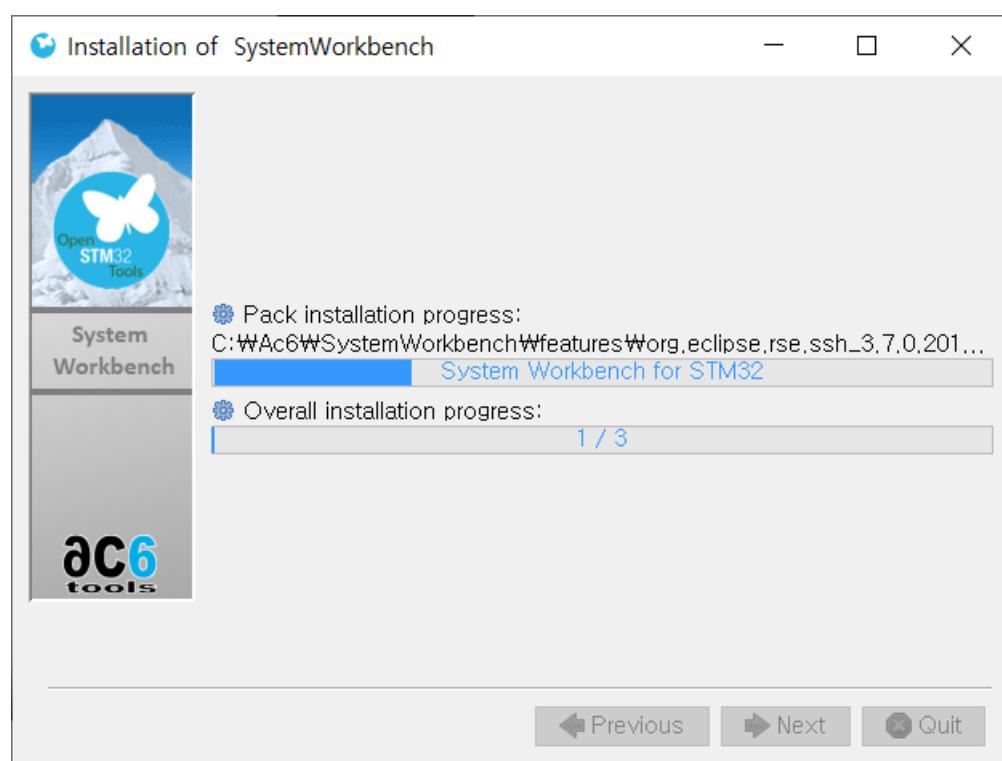
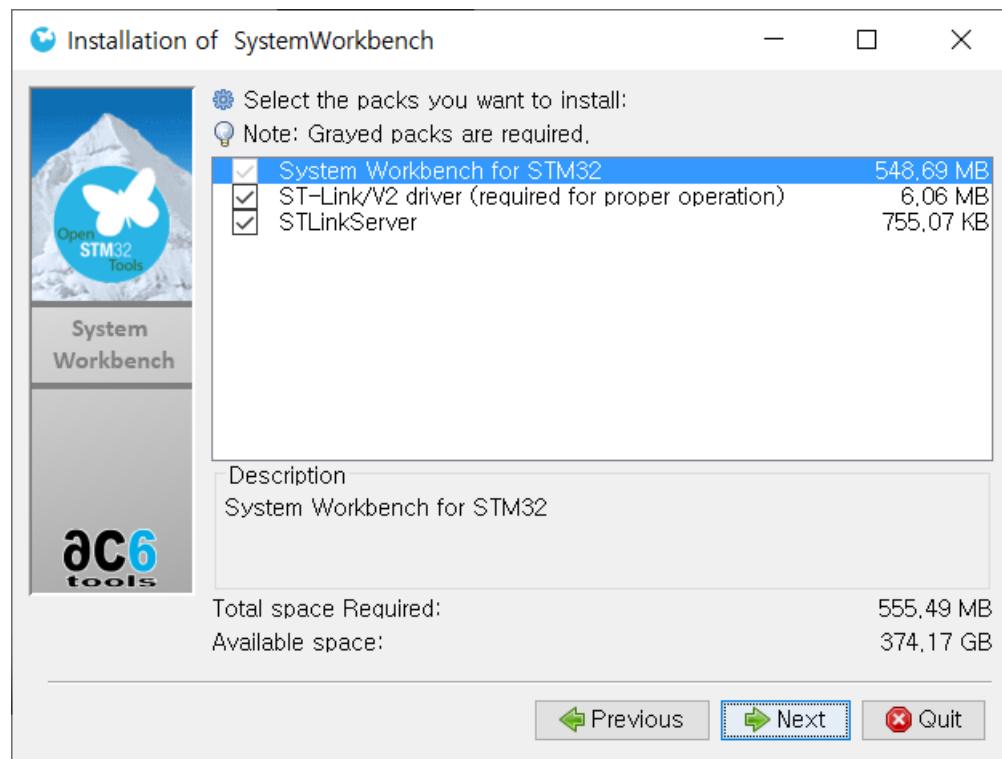
The Windows version is available for 32 and 64 bit systems. Note that we will need to install a device driver to communicate with the ST-Link debug probe, so you **must** select the installer that fits your system. Installing the 32 bit version on a 64 bit Windows system will **not** work. If you have problems downloading an executable file (.exe), try downloading and extracting the ZIP file. In both cases you are advised to also download the MD5 or SHA256 checksum to validate the integrity of your download.

- Latest Windows 64 bit installer, version v2.9, updated on Friday, April 12, 2019 at 16:41:04 CEST:
 - Installer: [install_sw4stm32_win_64bits-v2.9.exe](#)?
 - MD5 sum d5aa0fc2e30000ac000ac2rc0f04c in [install_sw4stm32_win_64bits-v2.9.exe.md5](#) ?
 - SHA256 sum 000cc9745b22e82ac51ec0b0880a56c8ee605d512d6ed85a6dd401f3ff7f22 in [install_sw4stm32_win_64bits-v2.9.exe.sha256](#) ?
 - Zipped installer: [install_sw4stm32_win_64bits-v2.9.zip](#)?
 - MD5 sum cd9e6feda96332a2237a40ceca4b1c9 in [install_sw4stm32_win_64bits-v2.9.zip.md5](#) ?
 - SHA256 sum 4aa85564ae9a8d1882647b9fea11c2de1ddb6230389a6db9e08a93028b5282b in [install_sw4stm32_win_64bits-v2.9.zip.sha256](#) ?
- The latest installer can always be retrieved from
 - Installer: [install_sw4stm32_win_64bits-latest.exe](#)?
 - MD5 sum of the installer: [install_sw4stm32_win_64bits-latest.exe.md5](#) ?
 - SHA256 sum of the installer: [install_sw4stm32_win_64bits-latest.exe.sha256](#) ?
 - Zipped installer: [install_sw4stm32_win_64bits-latest.zip](#)?
 - MD5 sum of the zip: [install_sw4stm32_win_64bits-latest.zip.md5](#) ?
 - SHA256 sum of the zip: [install_sw4stm32_win_64bits-latest.zip.sha256](#) ?
- Latest Windows 32 bit installer Version v2.9, updated on Friday, April 12, 2019 at 16:41:10 CEST:
 - Installer: [install_sw4stm32_win_32bits-v2.9.exe](#)?
 - MD5 sum a8d10d2bb72b2c9dd3dc03b626e16d28 in [install_sw4stm32_win_32bits-v2.9.exe.md5](#) ?
 - SHA256 sum 5ed29a52fe809e129b3b1494d7e4ac2b9755348973a0bf202577a2f0a398aa2 in [install_sw4stm32_win_32bits-v2.9.exe.sha256](#) ?
 - Zipped installer: [install_sw4stm32_win_32bits-v2.9.zip](#)?
 - MD5 sum 7e9499815447315e3b8f608ce36efab4 in [install_sw4stm32_win_32bits-v2.9.zip.md5](#) ?
 - SHA256 sum 757d654591f115823330f0e1598ab46535b7be871bccff2cf643a6ee25ae09 in [install_sw4stm32_win_32bits-v2.9.zip.sha256](#) ?
- The latest installer can always be retrieved from
 - Installer: [install_sw4stm32_win_32bits-latest.exe](#)?
 - MD5 sum of the installer: [install_sw4stm32_win_32bits-latest.exe.md5](#) ?
 - SHA256 sum of the installer: [install_sw4stm32_win_32bits-latest.exe.sha256](#) ?
 - Zipped installer: [install_sw4stm32_win_32bits-latest.zip](#)?
 - MD5 sum of the zip: [install_sw4stm32_win_32bits-latest.zip.md5](#) ?
 - SHA256 sum of the zip: [install_sw4stm32_win_32bits-latest.zip.sha256](#) ?

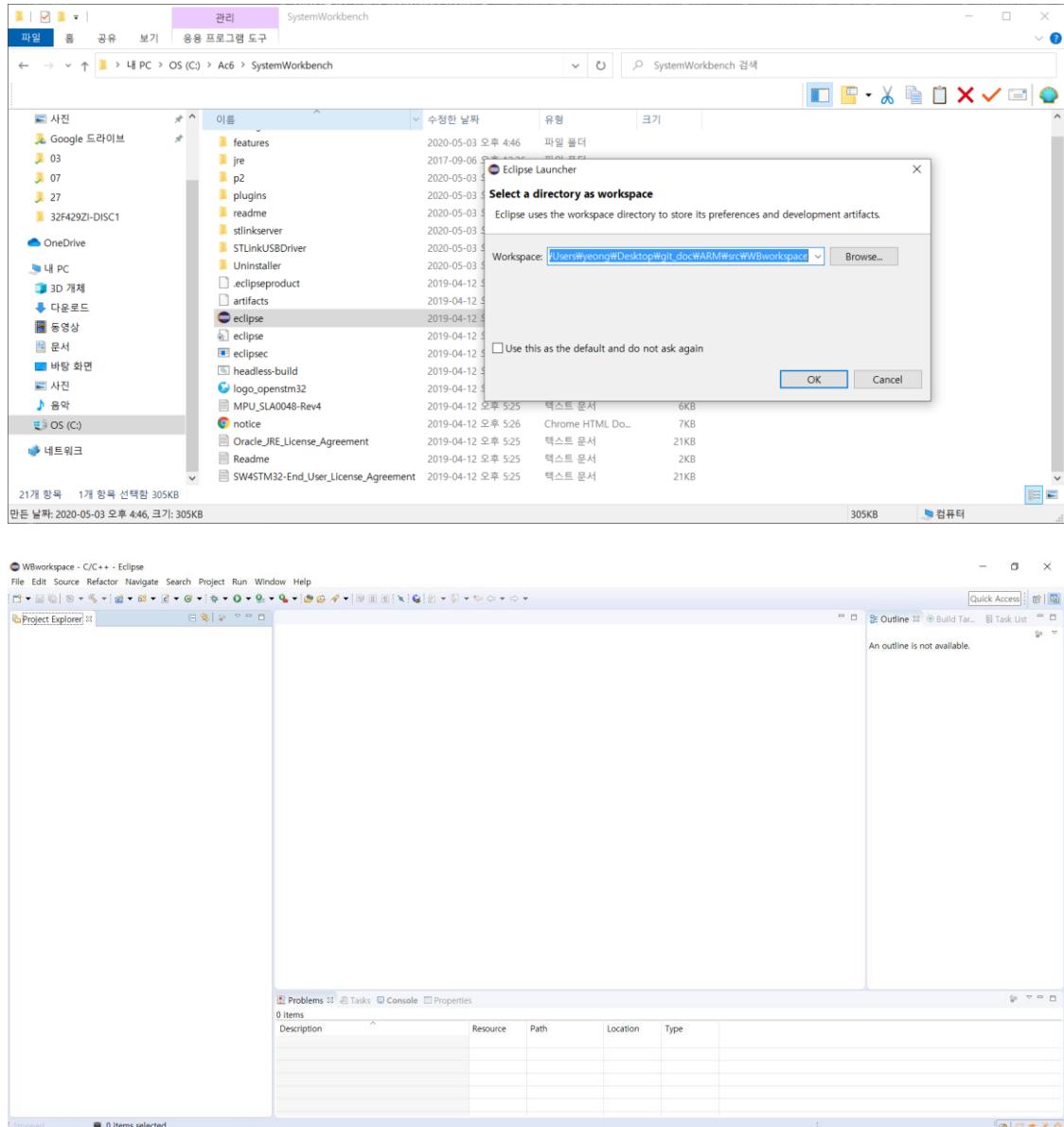
B. 그리고 설치파일을 실행시킨다



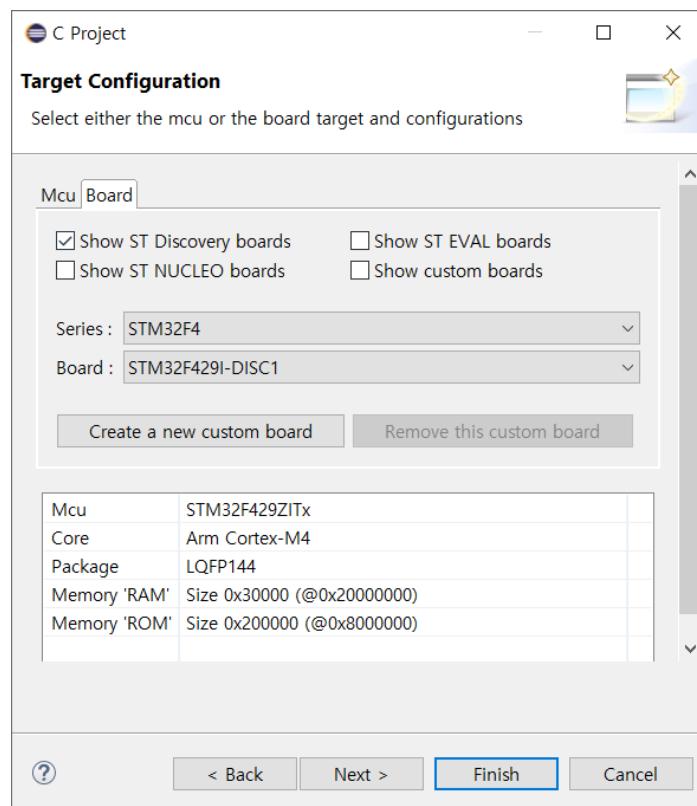
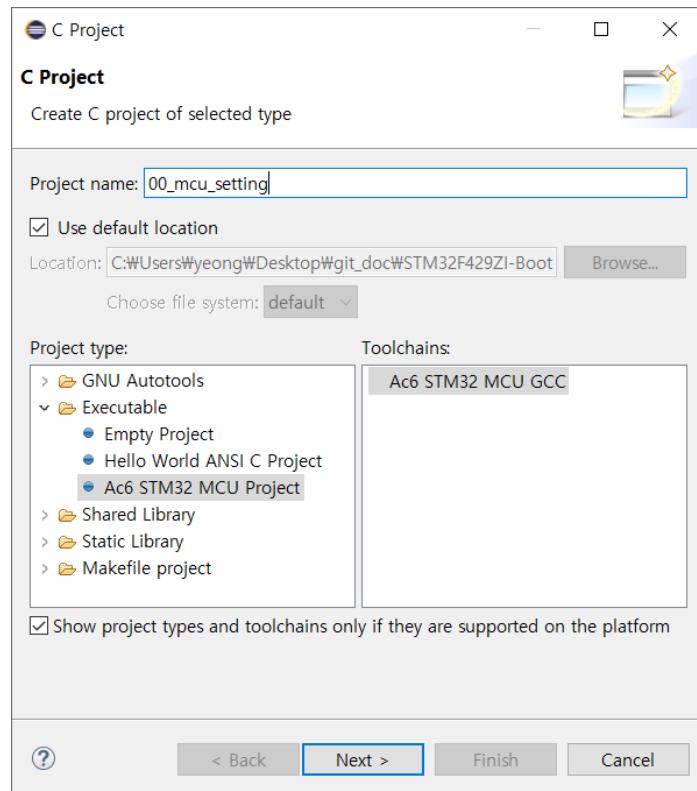
C. 모든 설정은 기본적으로 진행하도록 한다. 그러면 실행파일과 드라이버 설치가 이뤄질 것이다



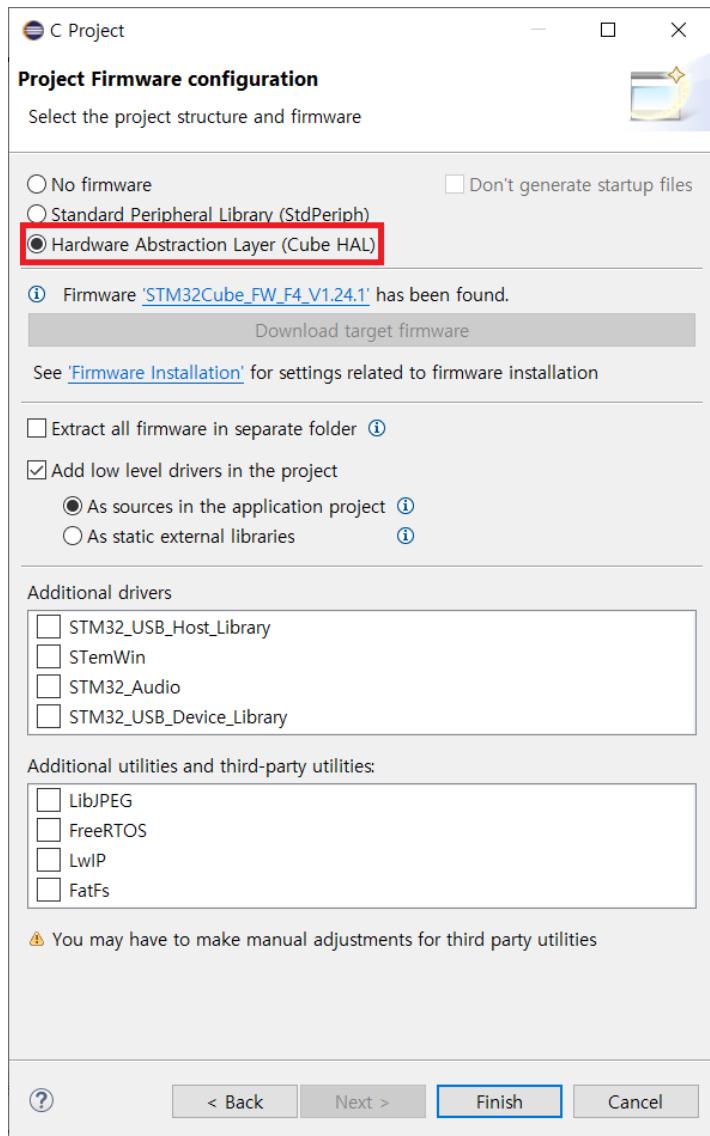
D. 설치가 모두 완료되면 기본 경로(C:/Ac6/SystemWorkbench)에 Eclipse Workbench 실행 파일이 생성되었을 것이다. 이를 실행시켜, 원하는 폴더에 지정한 후 실행시킨다. 처음 시작한다면 윈도우용 ARM 툴체인을 자동적으로 설치하게 된다. 끄지 말고 설치까지 기다려야 한다



E. 기본 프로젝트를 생성해 테스트할 것이다



그리고 HAL 라이브러리로 진행할 것이다



F. 간단한 Led 프로그램을 작성하도록 할 것이다. 그리고 Target 옵션에서 다운로드한 후 보드 상태를 Test 해보도록 한다

```
11 //  
12 #include "stm32f4xx.h"  
13 #include "stm32f429i_discovery.h"  
14  
15  
16 int main(void)  
17 {  
18     BSP_LED_Init(0);  
19     BSP_LED_Init(1);  
20  
21     BSP_LED_On(0);  
22     BSP_LED_On(1);  
23  
24     for(;;);  
25 }  
26 |
```

2. System Memory 부트로더

- 이전에 예제 프로그램을 실행하면서 Linker 설정에서 0x08000000번지에 프로그램을 다운로드 한다는 것을 알 수 있었다. 기본적으로 ST 마이크로컨트롤러는 Boot mode가 Main Flash memory로 되어있기 때문에 바로 다운로드된 부분부터 전원을 켜면 실행하게 된다. 그런데 ST MCU에는 Flash 외에도 SRAM과 System Memory(ROM) 옵션도 있다

2.4 Boot configuration

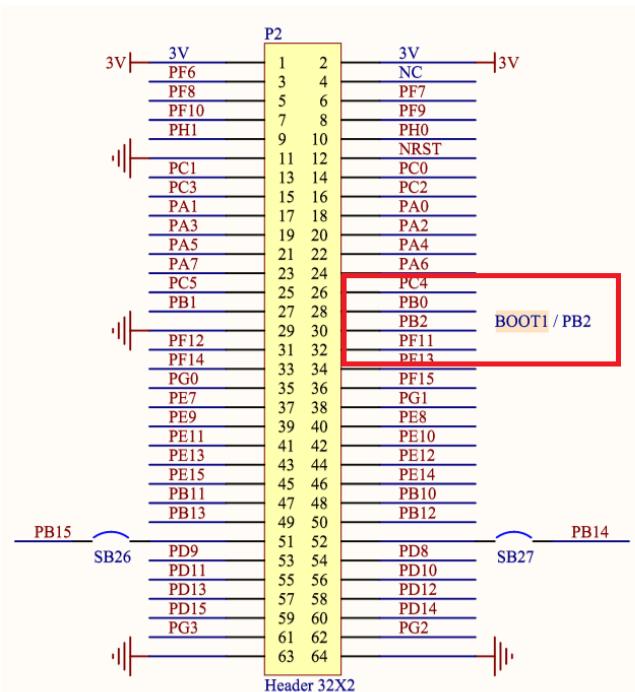
Due to its fixed memory map, the code area starts from address 0x0000 0000 (accessed through the ICode/DCode buses) while the data area (SRAM) starts from address 0x2000 0000 (accessed through the system bus). The Cortex®-M4 with FPU CPU always fetches the reset vector on the ICode bus, which implies to have the boot space available only in the code area (typically, Flash memory). STM32F4xx microcontrollers implement a special mechanism to be able to boot from other memories (like the internal SRAM).

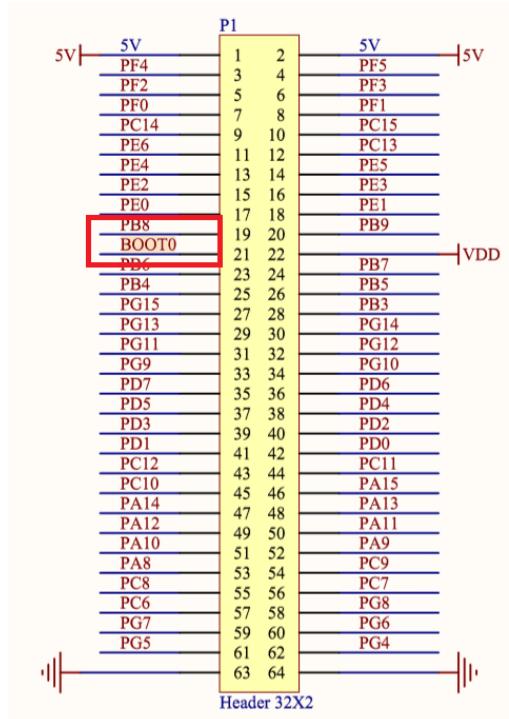
In the STM32F4xx, three different boot modes can be selected through the BOOT[1:0] pins as shown in [Table 2](#).

Table 2. Boot modes

Boot mode selection pins		Boot mode	Aliasing
BOOT1	BOOT0		
x	0	Main Flash memory	Main Flash memory is selected as the boot space
0	1	System memory	System memory is selected as the boot space
1	1	Embedded SRAM	Embedded SRAM is selected as the boot space

- 그리고 ST MCU에는 System memory ROM 부분에 내장 부트로더 코드가 이미 들어있다. 그래서 ST-Link가 없어도 ROM 부트로더를 이용해서도 다운로드를 할 수 있다. 그러려면 BOOTx핀을 System memory로 수정해야 한다. 다음 아래 사진을 보면서 핀을 맞출 수 있다





실제 보드를 뒤집으면 BOOT0은 표시되어 있을 것이다. BOOT1은 PB2를 찾으면 된다. 따라서 각각 5V, GND 핀과 연결하도록 한다

2.1 ST Native Bootloader

ROM 내장 부트로더에 대한 Application Note가 따로 제공되고 있다. 따라서 이에 대해서 파악해보려고 한다. 문서 링크는 [다음](#)과 같다



Introduction

The bootloader is stored in the internal boot ROM memory (system memory) of STM32 devices. It is programmed by ST during production. Its main task is to download the application program to the internal Flash memory through one of the available serial peripherals (USART, CAN, USB, I²C, SPI, etc.). A communication protocol is defined for each serial interface, with a compatible command set and sequences. This document applies to the products listed in [Table 1](#). They are referred as STM32 throughout the document.

This application note describes the supported peripherals and hardware requirements to be considered when using the bootloader of STM32 devices. However the specifications of the low-level communication protocol for each supported serial peripheral are documented in separate documents as referred in [Section 2: Related documents](#).

2.2 Flash loader demonstrator 설치 및 실행

- ST에서는 ROM 내장 부트로더뿐만 아니라 인터페이스를 할 수 있는 프로그램을 제공하고 있다. 그러면 하드웨어 BOOT 핀만 잘 설정한다면, 바로 내장 부트로더를 활용할 수 있다. 따라서 [ST 사이트](#)에서 다운로드를 받아보려고 한다. 사이트로 이동해보려고 한다

Get Software Featured Products Latest from ST

The STM32 Flash loader demonstrator (FLASHER-STM32) is a free software PC utility from STMicroelectronics, which runs on Microsoft® OSs and communicates through the RS232 with the STM32 system memory bootloader. To get an example of how to execute the device bootloader, refer to the STM32 microcontroller system memory boot mode Application note (AN2606). To get information about the USART protocol used in the STM32 bootloader, refer to the USART protocol used in the STM32 bootloader Application note (AN3155).

This software utility contains also a command line version and it is provided with Microsoft® Visual Studio 12 source code.

KEY FEATURES

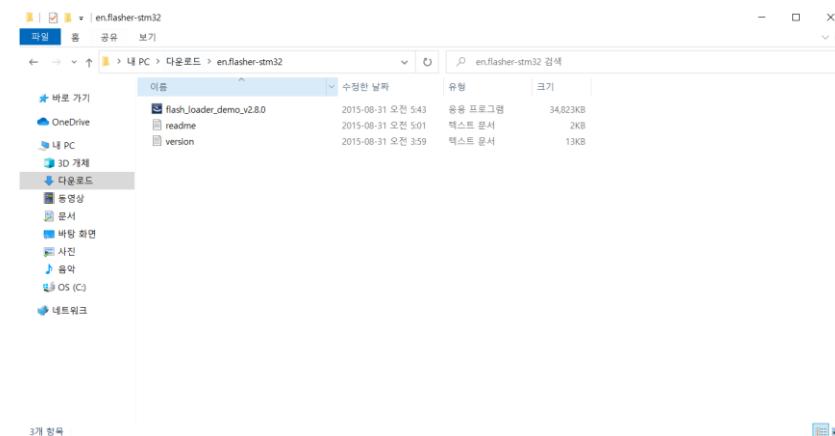
- UART system memory bootloader
- Flash programming utility with RS232
- It runs on Microsoft® Windows® OSs

- A. RS232를 통하여 OS와 Communication하는 PC utility다
- B. Window OS에서만 사용가능하며 UART-RS232 프로토콜을 통하여 flash가 가능하다
- C. 아래에서 프로그램을 다운로드 하도록 한다

Get Software

Part Number	General Description	Software Version	Supplier	Download
FLASHER-STM32	STM32 Flash loader demonstrator (UM0462)	2.8.0	ST	Get Software

- D. 받은 압축파일을 풀도록 한다



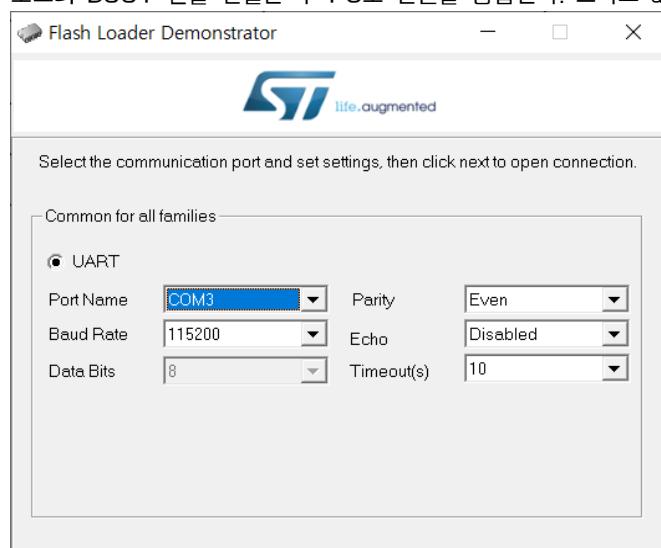
- E. 설치파일을 통해서 설치를 완료하면 다음과 같이 프로그램이 실행될 것이다. 프로그램 설명과 같이 UART 프로토콜만 지원하는 프로그램이다. 429Discovery보드는 전원 USB와 ST-Link가 같은 포트에서 USART1으로 연결되어 제공되기 때문에 이를 활용하려고 한다



Table 69. STM32F42xxx/43xxx configuration in system memory boot mode (continued)

Bootloader	Feature/Peripheral	State	Comment
USART1 bootloader	USART1	Enabled	Once initialized the USART1 configuration is: 8 bits, even parity and 1 Stop bit
	USART1_RX pin	Input	PA10 pin: USART1 in reception mode
	USART1_TX pin	Output	PA9 pin: USART1 in transmission mode

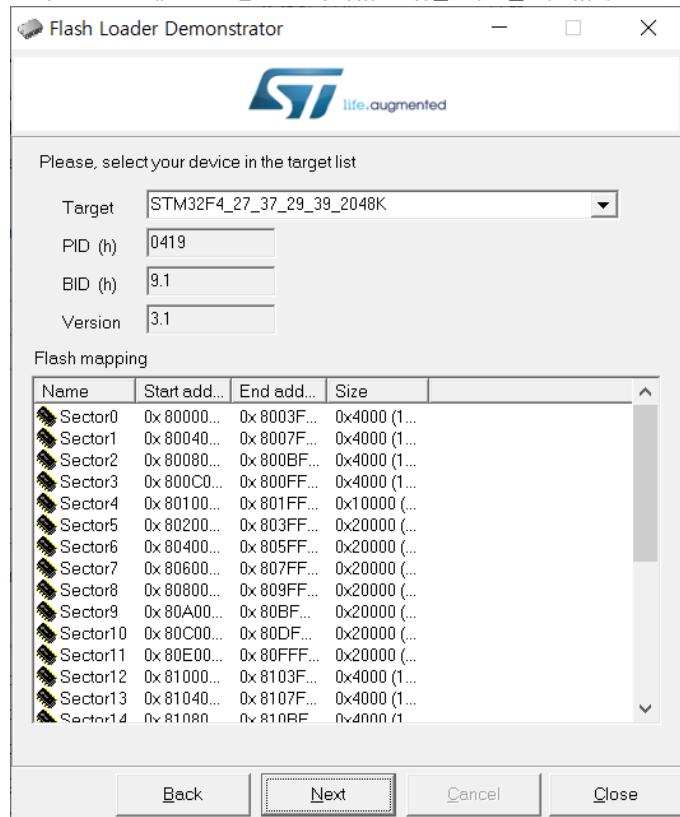
- F. 보드의 BOOT 핀을 연결한 후 PC로 전원을 공급한다. 그리고 demo 프로그램을 열어보도록 한다



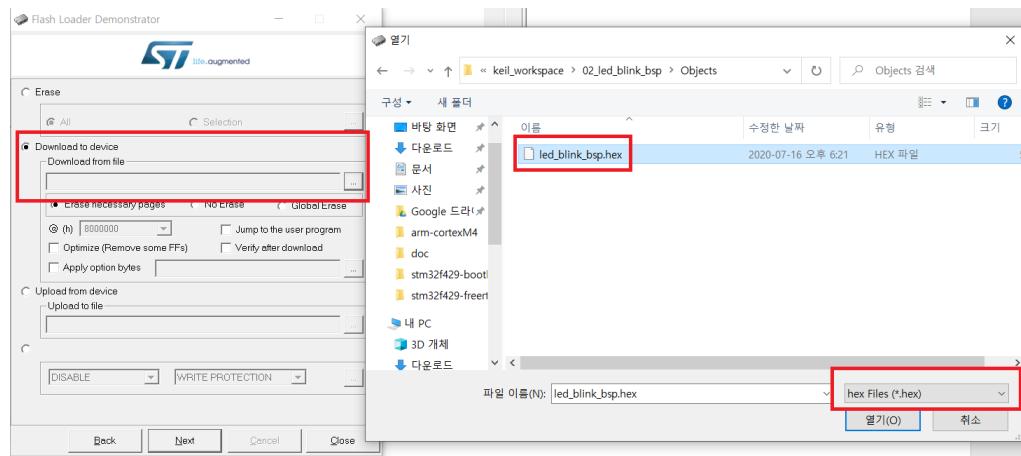
- G. 현재 연결된 COM PORT로 연결한 후에 다음(Next)을 누른다. BOOTX 핀들이 System Memory로 잘 꽂혀 있어야만 아래 화면이 나오게 된다



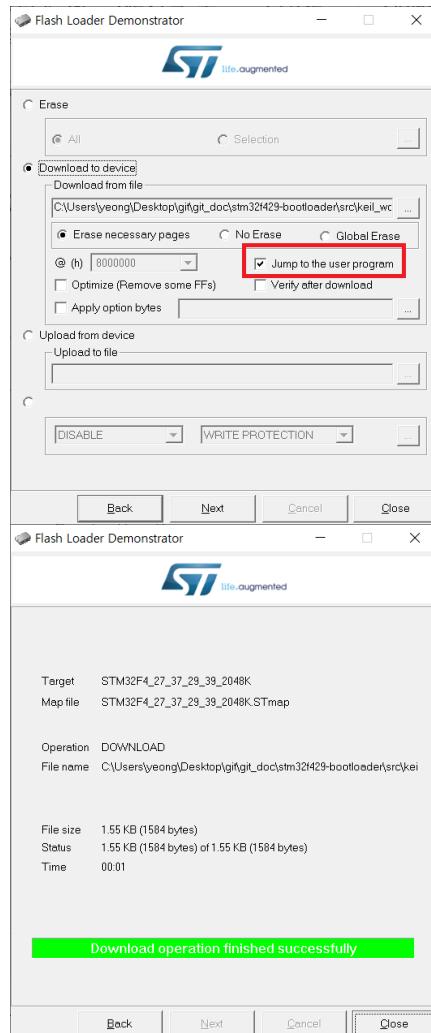
- H. 그리고 MCU에 대한 정보도 나와있는 것을 확인할 수 있다



- I. 그리고 계속 Next를 누르면 Erase 혹은 Download 여부를 묻게 된다. Download를 선택해서 아까 실습한 Keil 프로젝트 중 blink_bsp.hex 파일을 선택해서 다운로드 하려고 한다



다운로드 이후를 하더라도 BOOT 핀에 의해서 시스템 메모리에 머무르게 되는데, 다음 항목을 체크하면 바로 0x8000000번지로 점프를 해서 user 프로그램이 실행된다



다운로드가 성공적으로 된 것을 확인할 수 있다. 그리고 보드의 led가 켜 있는 것도 확인이 되었다

2.3 다른 UART로 시스템 부트로더 사용하기

USART1외에도 지원되는 인터페이스는 여러 개가 있는 것을 확인할 수 있다. 그 중 USART3을 사용하려고 한다. PB10과 11은 각각 TX, RX다. 보드에 다른 UART 모듈을 핀을 번갈아 연결하도록 한다

USART3 bootloader (on PB10/PB11)	USART3	Enabled	Once initialized the USART3 configuration is: 8 bits, even parity and 1 Stop bit
	USART3_RX pin	Input	PB11 pin: USART3 in reception mode
	USART3_TX pin	Output	PB10 pin: USART3 in transmission mode
USART3 bootloader (on PC10/PC11)	USART3	Enabled	Once initialized the USART3 configuration is: 8 bits, even parity and 1 Stop bit
	USART3_RX pin	Input	PC11 pin: USART3 in reception mode
	USART3_TX pin	Output	PC10 pin: USART3 in transmission mode

연결한 후 리셋을 누른 후 다시 프로그램을 열면 COM8이 생긴 것을 볼 수 있다



똑같이 COM3(UART1)와 같이 이용할 수 있다는 것을 확인할 수 있다

2.4 더 나아가기. Hex 파일 이해하기

Hex는 ASCII 텍스트 형식으로 이진 정보를 전달하는 파일 형식을 말한다. 이것은 MCU, EEPROM 등에 프로그래밍을 위해 많이 사용된다. 이들은 시리얼통신의 제어문자와 바이너리 데이터를 구분하기 위해서 HEX 포맷을 가지고 있다. 실제로 MCU 벤더 사인 ST에서의 Hex 파일로 프로그램을 다운로드를 하며, 다음과 같은 규칙을 가지고 있다

:	데이터 바이트 개수	오프셋	레코드 종류	데이터 바이트	체크섬
---	------------	-----	--------	---------	-----

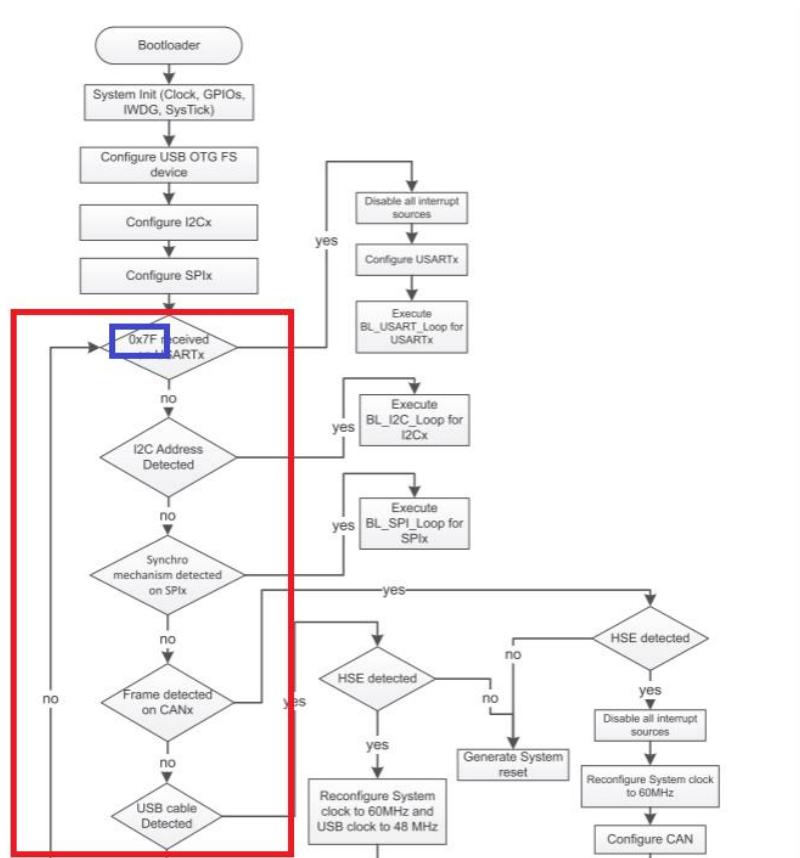
이전에 다룬 led_blink_bsp.hex 파일을 열어서 hex 파일을 분석해보려고 한다

```
I:020000040800F2
:100000006006002031020008390200083B020008A7
:100010003D0200083F020008410200080000000005
:10002000000000000000000000000000004302000883
:1000300045020008000000004702000849020008CD
:100040004B0200084B0200084B0200084B0200085C
:100050004B0200084B0200084B0200084B0200084C
:100060004B0200084B0200084B0200084B0200083C
:100070004B0200084B0200084B0200084B0200082C
:100080004B0200084B0200084B0200084B0200081C
:100090004B0200084B0200084B0200084B0200080C
:1000A0004B0200084B0200084B0200084B020008FC
```

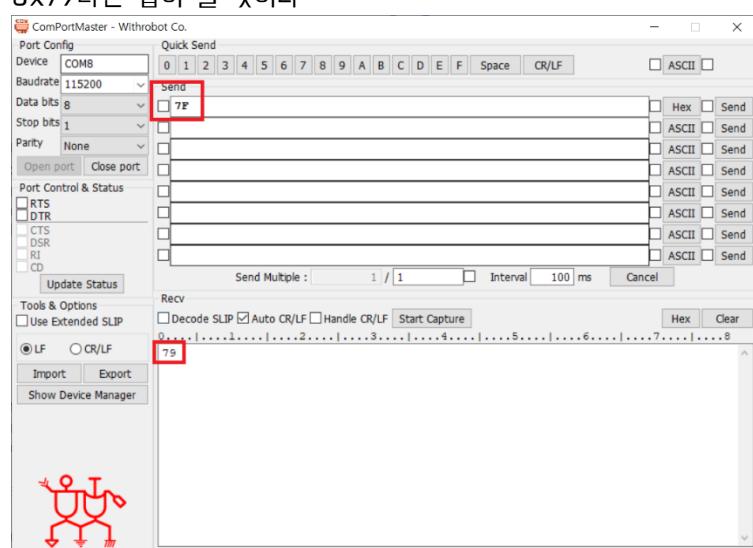
- A. 첫 시작은 콜론(:)으로 hex 바이너리 파일임을 알려주는 식별자로 hex값으로 0x3A다
 - B. 02는 2바이트를 의미하며 뒤에 0800이 이를 의미한다
 - C. 04는 뒤에 데이터를 쓸 때 기준 주소가 된다
 - D. 0000은 뒤의 기준 시작주소인 0800에 대한 오프셋 값이다. 현재는 없다는 것을 말한다
 - E. 그리고 F2는 모든 값을 더한 후 보수를 취한 값이다
 - F. 두번째 줄 레코드를 분석하면 10은 0x10 hex 값을 의미하며, 16바이트가 뒤에 있다고 말할 수 있다. 그리고 0000은 레코드 주소 오프셋 값을 의미한다
 - G. 그리고 뒤에 00은 뒤에 값들이 데이터라는 것을 의미한다
 - H. 0xA7 체크섬 값까지 총 16바이트가 나열된 것을 볼 수 있다
- 이를 설명한 이유는 고가의 장비가 없어도 UART로 hex 레코드 형식으로 시작 레코드 주소, 종료 레코드 그리고 데이터 값을 올바르게 전달한다면 펌웨어 업데이트가 크게 어렵지 않다는 것이다. 따라서 ST에서 제공하는 UART용 demo 프로그램도 hex를 지원하는데, 프로토콜 방식대로 읽어서 데이터를 쓰기 때문에 어렵지 않게 부트로더를 만들 수 있다는 것이다

2.5 시스템 부트로더 Flowchart

- 다음 링크에 시스템 메모리에 대한 BOOT 시퀀스를 보여주고 있다. 각 MCU에서 지원하는 주변장치들을 무한루프로 검사해서 시스템 메모리 부트로더를 실행시킨다



- 실제로 리셋을 한 이후에シリ얼 프로그램으로 0x7F을 보내도록 하자. 그러면 부트로더에서 0x79라는 답이 올 것이다



데이터시트에서는 ACK 신호를 0x79라고 표현하고 있다

Next, the code initializes the serial interface accordingly. Using this calculated baud rate, an acknowledge byte (0x79) is returned to the host, which signals that the STM32 is ready to receive commands.

NACK은 0x1F라고 한다

Each packet is either accepted (ACK answer) or discarded (NACK answer):

- ACK = 0x79
- NACK = 0x1F

부트로더 지원 보율 범위는 다음과 같다. 1200 - 115200까지 지원이 된다. 보통 익히 알고 있는 **9600, 115200 보율을 사용하면 된다**

2.1 Minimum baud rate

The lowest tested baud rate (B_{Low}) is 1200. Baud rates below B_{Low} cause SysTick timer overflows. In this event, USARTTx will not be correctly initialized.

2.2 Maximum baud rate

B_{High} is the highest baud rate for which the deviation does not exceed the limit. All baud rates between B_{Low} and B_{High} are below the deviation limit.

The highest tested baud rate (B_{High}) is 115200.

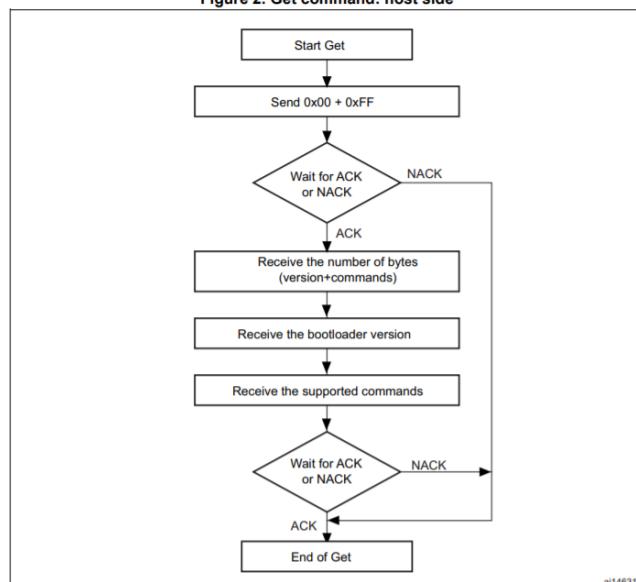
A. Get 명령

Get 명령은 MCU에 내장된 부트로더 버전과 함께 해당 버전의 부트로더에서 지원하는 명령어들을 차례대로 확인할 수 있다. Get을 실행하는 방법은 연속적으로 MCU에 0x00과 0xFF 명령을 전달하면 된다. 그러면 ACK 신호와 함께 해당 정보가 보내진다

Get command

The Get command allows the user to get the version of the bootloader and the supported commands. When the bootloader receives the Get command, it transmits the bootloader version and the supported command codes to the host, as described in [Figure 2](#).

Figure 2. Get command: host side



그리고 명령에 대한 답은 ACK + 보낼 데이터 바이트 + 데이터 + ACK가 올 것이다

The STM32 sends the bytes as follows:

Byte 1:	ACK
Byte 2:	N = 11 = the number of bytes to follow – 1 except current and ACKs.
Byte 3:	Bootloader version (0 < Version < 255), example: 0x10 = Version 1.0
Byte 4:	0x00 – Get command
Byte 5:	0x01 – Get Version and Read Protection Status
Byte 6:	0x02 – Get ID
Byte 7:	0x11 – Read Memory command
Byte 8:	0x21 – Go command
Byte 9:	0x31 – Write Memory command
Byte 10:	0x43 or 0x44 – Erase command or Extended Erase command (exclusive commands)
Byte 11:	0x63 – Write Protect command
Byte 12:	0x73 – Write Unprotect command

AN3155 Rev 9



Command set

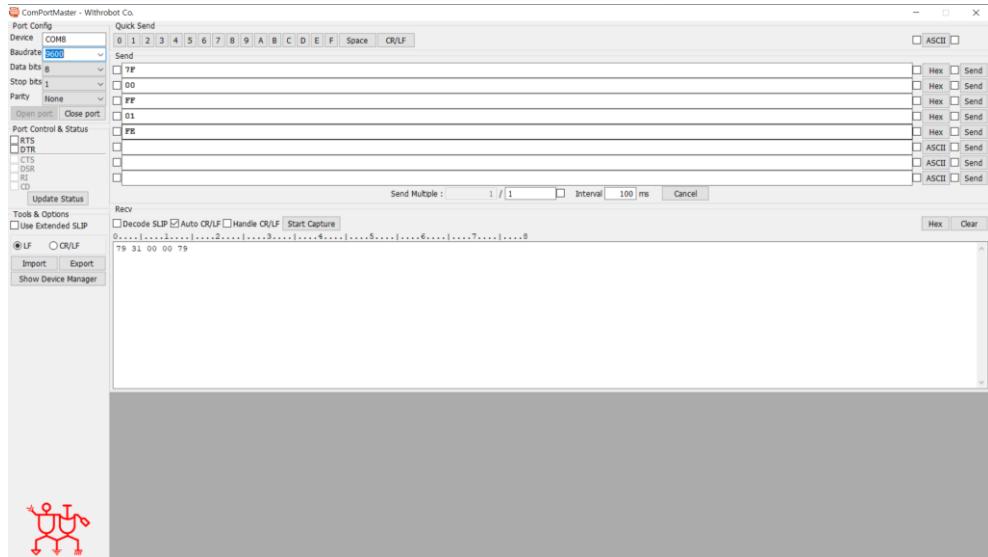
AN

Byte 13:	0x82	– Readout Protect command
Byte 14:	0x92	– Readout Unprotect command
Last byte (15):	ACK	

아래를 토대로 0x01 명령으로 Get Version과 Read Protection Status를 알아보려고 한다

B. Get Version & Read Protection Status command

터미널 프로그램으로 0x01 + 0xFE 명령을 보내도록 한다



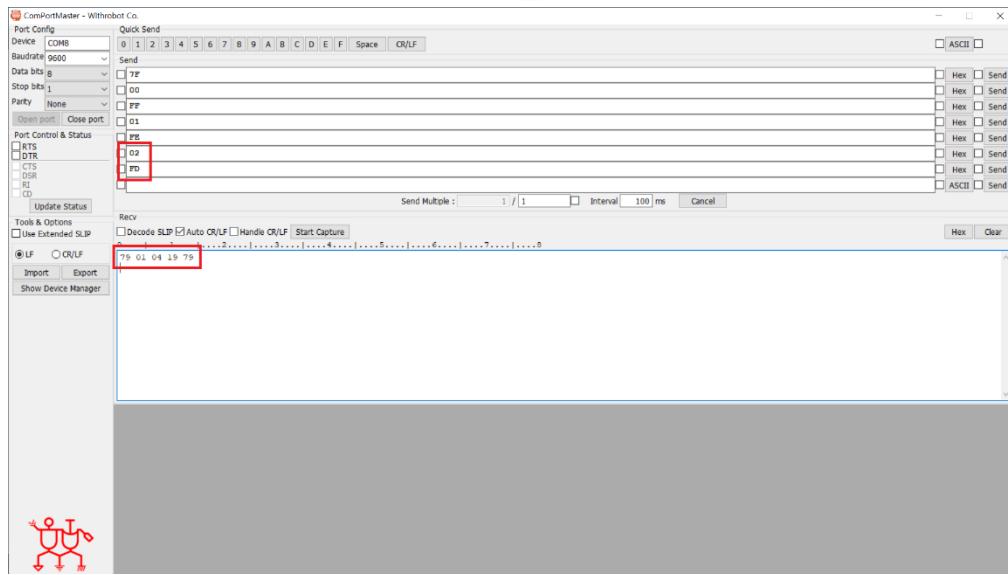
The STM32 sends the bytes as follows:

- Byte 1: ACK
- Byte 2: Bootloader version ($0 < \text{Version} \leq 255$), example: 0x10 = Version 1.0
- Byte 3: Option byte 1: 0x00 to keep the compatibility with generic bootloader protocol
- Byte 4: Option byte 2: 0x00 to keep the compatibility with generic bootloader protocol
- Byte 5: ACK

부트로더 버전은 31이라는 값에서 3.1로 추측이 가능하다. 그리고 option byte는 현재 크게 건드린 부분이 없기 때문에 0x00을 유지하게 된다. option byte에 대해서는 자세하게 다뤄보려고 한다. 그리고 마지막에 ACK를 전달받게 된다

C. Get ID command

터미널 프로그램으로 0x02 + 0xFD 명령을 보내도록 한다



The STM32 device sends the bytes as follows:

Byte 1: ACK

Byte 2: N = the number of bytes – 1 (N = 1 for STM32), except for current byte and ACKs.

Bytes 3-4: PID⁽¹⁾ byte 3 = 0x04, byte 4 = 0xXX

Byte 5: ACK

1. PID stands for product ID. Byte 1 is the MSB and byte 2 the LSB of the ID.

뒤에 2바이트가 나오는데 1을 뺀 값이 byte2에 포함되어서 적하게 된다. 따라서 N=1이 되며, STM에서는 1이라고 데이터시트에 명시되어 있다. 그리고 0x04 0x19는 아래 429 칩의 PID를 의미한다

Table 122. Bootloader device-dependent parameters (continued)

STM32 series	Device	PID	BL ID	RAM memory	System memory	
F3	STM32F373xx	0x432	0x41	0x20001400 - 0x20007FFF	0x1FFFD800 - 0x1FFFF7FF	
	STM32F378xx		0x50	0x20001000 - 0x20007FFF		
	STM32F302xB(C)/303xB(C)	0x422	0x41	0x20001400 - 0x20009FFF		
	STM32F358xx		0x50			
	STM32F301xx/302x4(6/8)	0x439	0x40	0x20001800 - 0x20003FFF		
	STM32F318xx		0x50			
	STM32F303x4(6/8)/334xx/328xx	0x438	0x50	0x20001800 - 0x20002FFF		
	STM32F302xD(E)/303xD(E)	0x446	0x40	0x20001800 - 0x2000FFFF		
	STM32F398xx	0x446	0x50	0x20001800 - 0x2000FFFF		
F4	STM32F40xxx/41xxx	0x413	0x31	0x20002000 - 0x2001FFFF	0x1FFF0000 - 0x1FFF77FF	
			0x90	0x20003000 - 0x2001FFFF		
	STM32F42xxx/43xxx	0x419	0x70	0x20003000 - 0x2002FFFF		
			0x91			
	STM32F401xB(C)	0x423	0xD1	0x20003000 - 0x2000FFFF		
	STM32F401xD(E)	0x433	0xD1	0x20003000 - 0x20017FFF		
	STM32F410xx	0x458	0xR1	0x20003000 - 0x20007FFF		

BLID는 메모리에 저장되어 있는 값을 읽어야 한다. 이는 시스템 메모리쪽에 정보가 저장되어 있어서 이를 읽을 수 있다

Table 3. Memory mapping vs. Boot mode/physical remap in STM32F405xx/07xx and STM32F415xx/17xx

Addresses	Boot/Remap in main Flash memory	Boot/Remap in embedded SRAM	Boot/Remap in System memory	Remap in FSMC
0x2001 C000 - 0x2001 FFFF	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)	SRAM2 (16 KB)
0x2000 0000 - 0x2001 BFFF	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)	SRAM1 (112 KB)
0xFFFF 0000 - 0xFFFF 77FF	System memory	System memory	System memory	System memory
0x0810 0000 - 0x0FFF FFFF	Reserved	Reserved	Reserved	Reserved
0x0800 0000 - 0x080F FFFF	Flash memory	Flash memory	Flash memory	Flash memory
0x0400 0000 - 0x07FF FFFF	Reserved	Reserved	Reserved	FSMC bank 1 NOR/PSRAM 2 (128 MB Aliased)
0x0000 0000 - 0x000F FFFF ⁽¹⁾⁽²⁾	Flash (1 MB) Aliased	SRAM1 (112 KB) Aliased	System memory (30 KB) Aliased	FSMC bank 1 NOR/PSRAM 1 (128 MB Aliased)

				I2C (V1.0)
F4	STM32F42xxx/43xxx	USART1/USART3/ CAN2 /DFU (USB Device FS) /I2C1	0x70	0x1FFF76DE
		USART1/USART3/ CAN2 / DFU (USB Device FS) / I2C1/I2C2/I2C3/SPI1/ SPI2/ SPI4	0x91	0x1FFF76DE
	STM32F401xB(C)	USART1/USART2/ DFU (USB Device FS)/ I2C1/I2C2/I2C3/ SPI1/SPI2/ SPI3	0xD1	0x1FFF76DE
	STM32F401xD(E)	USART1/USART2/ DFU (USB Device FS)/ I2C1/I2C2/I2C3/ SPI1/SPI2/ SPI3	0xD1	0x1FFF76DE
	STM32F410xx	USART1/USART2/ I2C1/I2C2/I2C4 SPI1/SPI2	0xB1	0x1FFF76DE
	STM32F411xx	USART1/USART2/ DFU (USB Device FS)/ I2C1/I2C2/I2C3/ SPI1/SPI2/ SPI3	0xD0	0x1FFF76DE
	STM32F412xx	USART1/USART2/ USART3/CAN2/ DFU (USB Device FS)/ I2C1/I2C2/I2C3/I2C4/ SPI1/SPI3/SPI4	0x91	0x1FFF76DE
		USART1/USART2/ USART3/CAN2/		USART (V3.1) CAN (V2.0)

91과 70을 나누는 기준은 데이터시트에 설명되어 있다. 지원되는 프로토콜로 앞자리를 나눈 다음 부트로더 버전으로 뒤의 자리를 구성한다

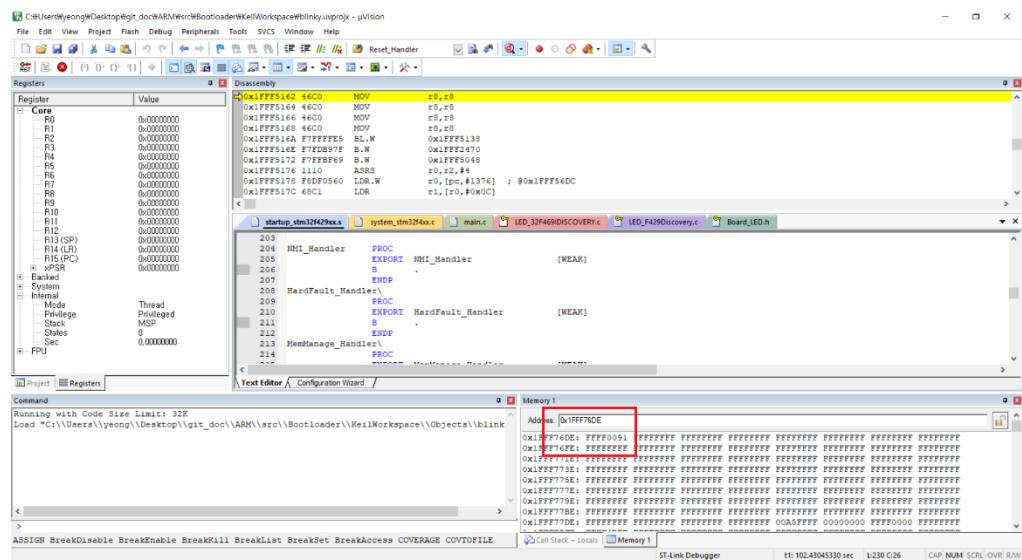
For a given STM32 device, the bootloader is identified by means of the:

1. **Bootloader (protocol) version:** version of the serial peripheral (USART, CAN, USB, etc.) communication protocol used in the bootloader. This version can be retrieved using the bootloader Get Version command.
2. **Bootloader identifier (ID):** version of the STM32 device bootloader, coded on one byte in the **0xXY** format, where:
 - **X** specifies the embedded serial peripheral(s) used by the device bootloader:
 - X = 1: one USART is used
 - X = 2: two USARTs are used
 - X = 3: USART, CAN and DFU are used
 - X = 4: USART and DFU are used
 - X = 5: USART and I²C are used
 - X = 6: I²C is used
 - X = 7: USART, CAN, DFU and I²C are used
 - X = 8: I²C and SPI are used
 - X = 9: USART, CAN, DFU, I²C and SPI are used**
 - X = 10: USART, DFU and I²C are used**
 - X = 11: USART, I²C and SPI are used**
 - X = 12: USART and SPI are used**
 - X = 13: USART, DFU, I²C and SPI are used**
 - **Y** specifies the device bootloader version

Let us take the example of a bootloader ID equal to 0x10. This means that it is the first version of the device bootloader that uses only one USART.

The bootloader ID is programmed in the last byte address - 1 of the device system memory and can be read by using the bootloader "Read memory" command or by direct access to the system memory via JTAG/SWD.

The table below provides identification information about the bootloaders embedded in STM32 devices.



그리고 실제로 읽었을 때 91이라는 값을 얻을 수 있다. 현재 부트로더는 SPI까지 지원한다는 것을 알 수 있다

D. Read Memory command

Read 명령은 지정한 메모리 번지에 대해서 값을 읽는 것이다. 읽을 수 있는 메모리는 이미 정해져 있고 Flash, RAM, System, Data 모두 가능한 것으로 볼 수 있다

The table below lists the valid memory area depending on the bootloader commands.

Table 5. Supported memory area by Write, Read, Erase and Go Commands

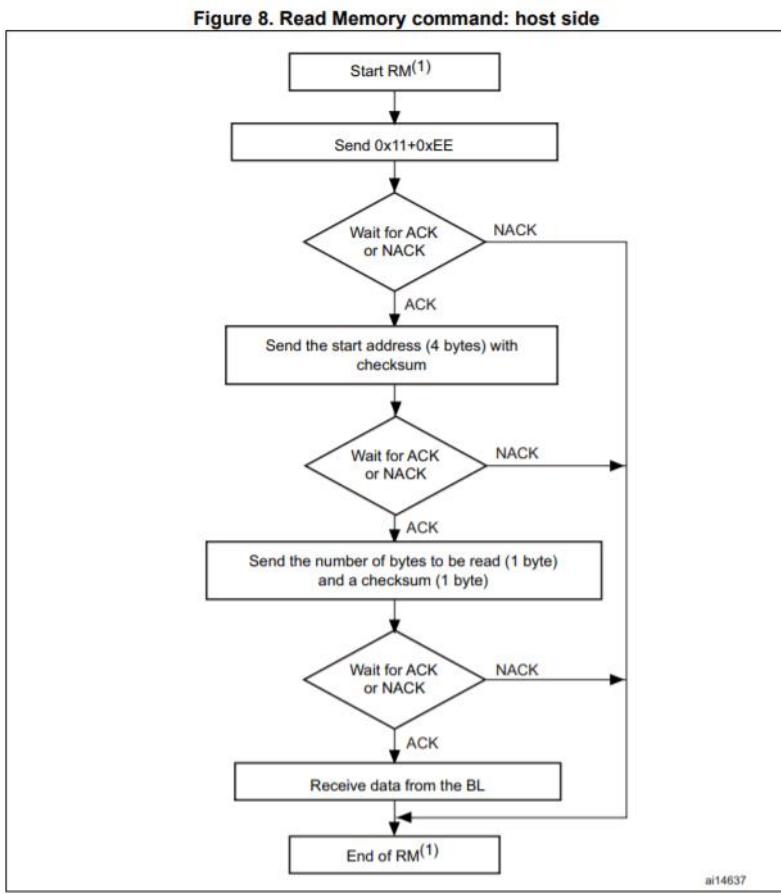
Memory Area	Write command	Read command	Erase command	Go command
Flash	Supported	Supported	Supported	Supported
RAM	Supported	Supported	Not supported	Supported
System Memory	Not supported	Supported	Not supported	Not supported
Data Memory	Supported	Supported	Not supported	Not supported
OTP Memory	Supported	Supported	Not supported	Not supported

When the bootloader receives the Read Memory command, it transmits the ACK byte to the application. After the transmission of the ACK byte, the bootloader waits for an address (four bytes, byte 1 is the address MSB and byte 4 is the LSB) and a checksum byte, then it checks the received address. If the address is valid and the checksum is correct, the bootloader transmits an ACK byte, otherwise it transmits a NACK byte and aborts the command.

When the address is valid and the checksum is correct, the bootloader waits for the number of bytes to be transmitted – 1 (N bytes) and for its complemented byte (checksum). If the checksum is correct it then transmits the needed data ((N + 1) bytes) to the application, starting from the received address. If the checksum is not correct, it sends a NACK before aborting the command.

반을 데이터는 N을 입력하면 N+1 바이트가 보내지는 것으로 보인다

그리고 다음 flow대로 진행이 된다. 여기서 BL ID를 읽어오는 과정을 해보려고 한다

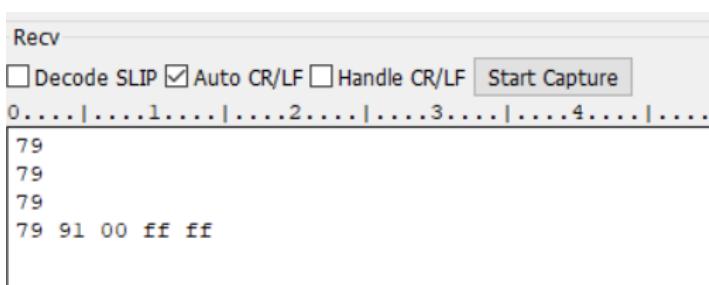


1. RM = Read Memory.

다음 메모리를 읽을 것이다. 아래 데이터 순서대로 보내도록 한다. 현재는 시스템 메모리 부트로더에 있다고 가정을 한다

STM32F42xxx/43xxx	USART1/USART3/ CAN2 / DFU (USB Device FS) / I2C1/I2C2/I2C3/SPI1/ SPI2/ SPI4	0x91	0xFFFF76DE	I2C (V1.0)
-------------------	---	------	------------	------------

0x11 0xEE 0x1F 0xFF 0x76 0xDE 0x48(체크섬) 0x03(4개 데이터) 0x7C(0x03 보수)



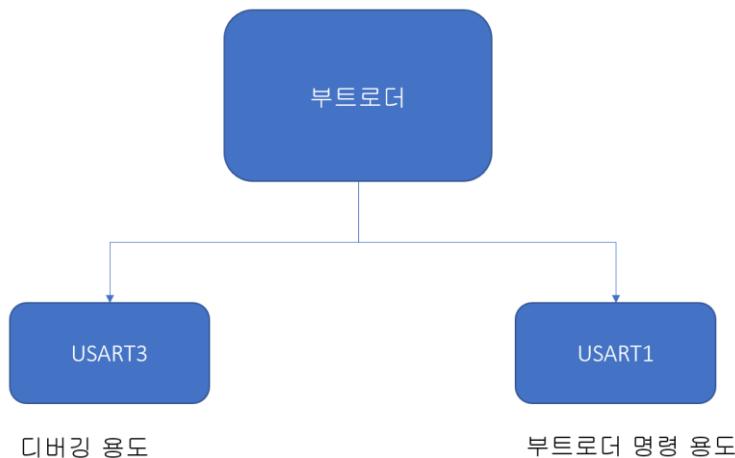
그럼 다음 데이터가 나오면서 현재 BL ID가 0x91이라는 것을 알 수 있다

3. Custom 부트로더 작성

3.1 Block Diagram

기존의 ROM 네이티브 부트로더를 참고해서 커스텀 부트로더를 만드려고 한다. 최종적으로 python으로 만든 PC 프로그램과 통신해서 PC에 있는 bin 파일을 UART로 전달해서 flash를 하는 것을 목적으로 하고 있다

대략적인 UART 시스템 블록 다이어그램은 다음과 같다



그리고 프로그램의 시작은 아무 하드웨어 연결없이 부트로더에서 하려고 하기 때문에 Flash 영역의 0, 1 섹터를 부트로더로 잡을 것이다. 나머지 영역은 user program을 다운로드 할 것이다. 따라서 버튼을 눌렀을 때는 그대로 부트로더를 실행하고, 그렇지 않다면 PC처럼 user program을 실행하는 구조다

RM0090

Embedded Flash memory interface

Table 6. Flash module - 2 Mbyte dual bank organization (STM32F42xxx and STM32F43xxx)

Block	Bank	Name	Block base addresses	Size
Main memory	Bank 1	Sector 0	0x0800 0000 - 0x0800 3FFF	16 Kbytes
		Sector 1	0x0800 4000 - 0x0800 7FFF	16 Kbytes
		Sector 2	0x0800 8000 - 0x0800 BFFF	16 Kbytes
		Sector 3	0x0800 C000 - 0x0800 FFFF	16 Kbytes
		Sector 4	0x0801 0000 - 0x0801 FFFF	64 Kbytes
		Sector 5	0x0802 0000 - 0x0803 FFFF	128 Kbytes
		Sector 6	0x0804 0000 - 0x0805 FFFF	128 Kbytes
		-	-	-
		-	-	-
		-	-	-
		Sector 11	0x080E 0000 - 0x080F FFFF	128 Kbytes
		Sector 12	0x0810 0000 - 0x0810 3FFF	16 Kbytes
		Sector 13	0x0810 4000 - 0x0810 7FFF	16 Kbytes
		Sector 14	0x0810 8000 - 0x0810 BFFF	16 Kbytes
		Sector 15	0x0810 C000 - 0x0810 FFFF	16 Kbytes
		Sector 16	0x0811 0000 - 0x0811 FFFF	64 Kbytes

실제로 내장 부트로더가 있는 ROM 시스템 메모리는 약 30K로, Flash에 다운로드 할 부트로더는 총 2섹터를 잡았기 때문에 32K다. 따라서 다양한 프로토콜을 지원하는 ROM 부트로더에 비해서는 충분한 용량으로 짐작이 된다

최종적으로 부트로더 프로그램 구조는 ST에서 제공하는 UART 프로토콜 [문서](#)를 참조해서, 이를 변형해서 가져갈 것이다. 아래는 ST에서 설계한 시스템 메모리 부트로더의 UART 프로토콜이다

Figure 1. Bootloader for STM32 with USART

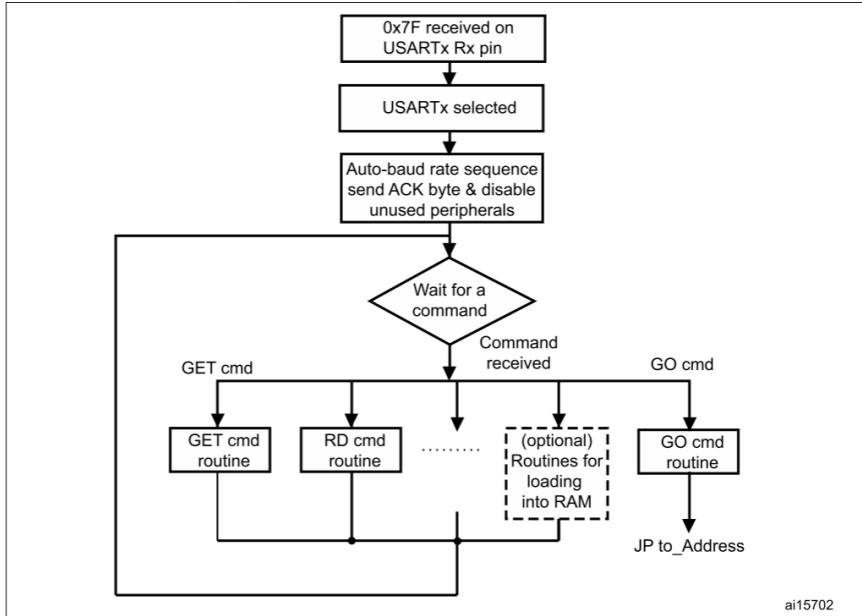


Table 2. USART bootloader commands

Command ⁽¹⁾	Command code	Command description
Get ⁽²⁾	0x00	Gets the version and the allowed commands supported by the current version of the bootloader.
Get Version & Read Protection Status ⁽²⁾	0x01	Gets the bootloader version and the Read Protection status of the Flash memory.
Get ID ⁽²⁾	0x02	Gets the chip ID.
Read Memory ⁽³⁾	0x11	Reads up to 256 bytes of memory starting from an address specified by the application.
Go ⁽³⁾	0x21	Jumps to user application code located in the internal Flash memory or in the SRAM.
Write Memory ⁽³⁾	0x31	Writes up to 256 bytes to the RAM or Flash memory starting from an address specified by the application.
Erase ⁽³⁾⁽⁴⁾	0x43	Erases from one to all the Flash memory pages.
Extended Erase ⁽³⁾⁽⁴⁾	0x44	Erases from one to all the Flash memory pages using two byte addressing mode (available only for v3.0 USART bootloader versions and above).
Write Protect	0x63	Enables the write protection for some sectors.
Write Unprotect	0x73	Disables the write protection for all Flash memory sectors.
Readout Protect	0x82	Enables the read protection.
Readout Unprotect ⁽²⁾	0x92	Disables the read protection.

- A. MCU는 리셋 후 리셋 핸들러를 거쳐서 메인 함수로 올 것이다. 메인 함수에서는 클럭, UART, CRC 등 주변장치 초기화 설정을 마친 후 버튼이 눌린 지 체크를 한다. 눌렸다면 HOST 명령을 기다릴 것이고, 아니라면 섹터 2의 user 프로그램을 실행시킨다
- B. 커맨드 내용은 위의 ST 문서를 토대로 짤 것이다. 그리고 CRC 기반으로 정해진 메뉴에서만 부트로더 통신을 실행한다

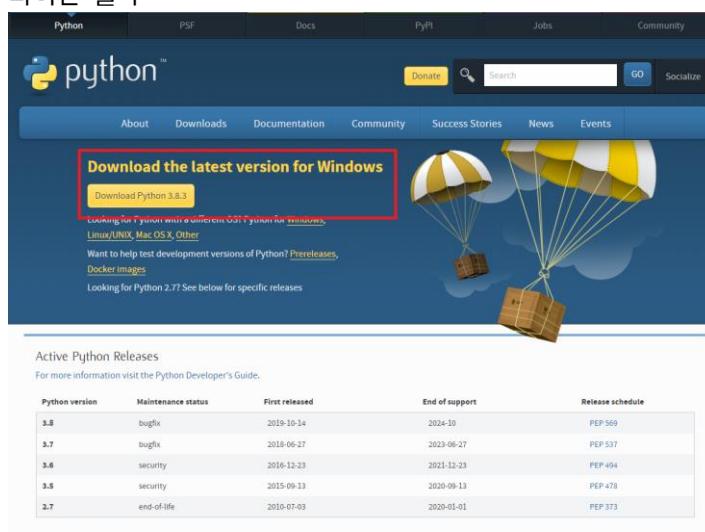
4. PC Host 통신 프로그램 작성 [Python]

- 타겟 부트로더와 통신하기 위해서는 PC 통신 프로그램이 반드시 필요하다. 그리고 타겟의 명령 줄 flash는 직접 파일 전달해야 되고, 명령마다 정해진 길이만큼 타겟에 보내야 하는 역할을 담당해야 한다. 따라서 Host 프로그램을 타겟과 마찬가지로 명령에 따라서 소스코드를 구조적으로 짜야 한다
- 그리고 Python의 장점은 스크립트 언어로 동작하고, 인터프리터로 해석하고 실행하기 때문에 OS 종속없이 프로그램을 짤 수 있다는 점이다. 컴파일러와 기계어로 실행되는 C로 serial 프로그램을 작성한다면, Window, Linux, Mac 모두 구분해서 작성해야 한다

4.1 python 설치

위에서 언급됐던 장점 때문에 python을 사용해서 host 프로그램을 작성하려고 한다. 따라서 먼저 파이썬 공식 홈페이지로 가서 3.x 버전 이상으로 다운로드 하도록 한다

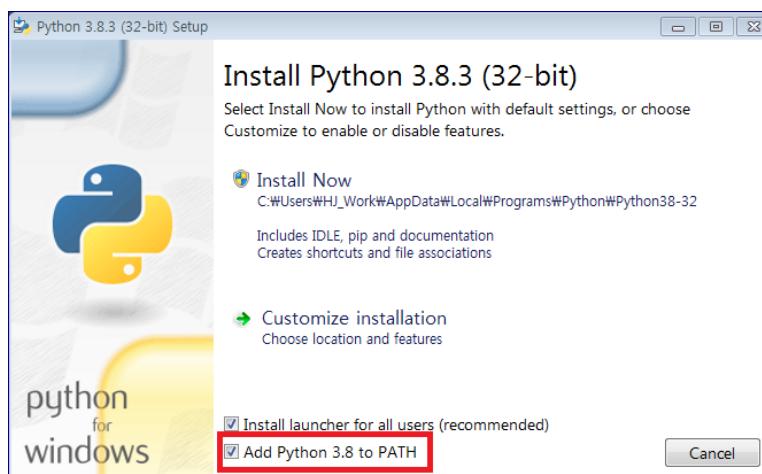
A. 파이썬 설치



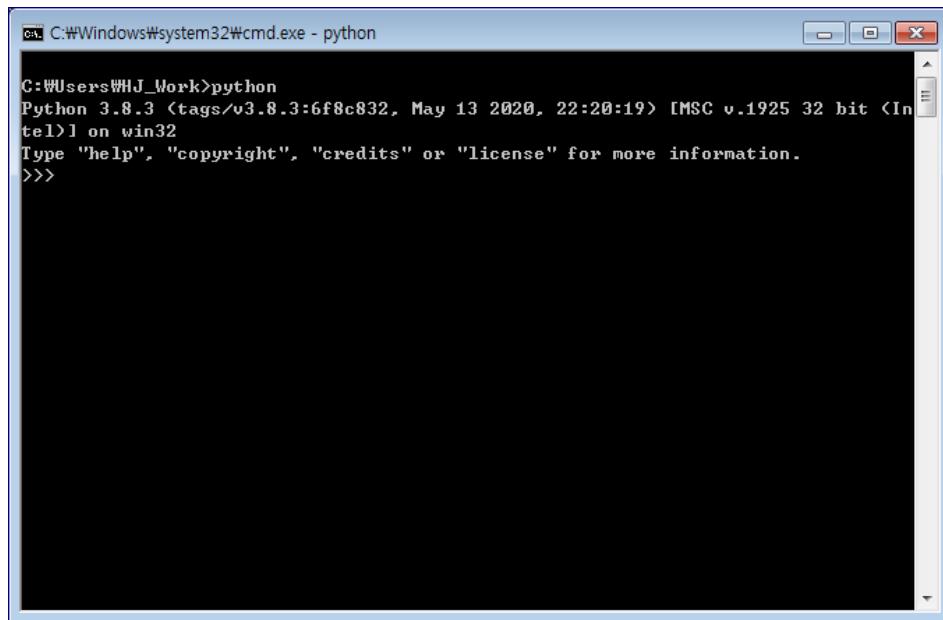
The screenshot shows the Python official website's main page. A large yellow button labeled "Download the latest version for Windows" is highlighted with a red box. Below it, there's a section for "Active Python Releases" with a table showing the following information:

Python version	Maintenance status	First released	End of support	Release schedule
3.8	bugfix	2019-10-14	2024-10	PEP 569
3.7	bugfix	2018-06-27	2023-06-27	PEP 537
3.6	security	2018-12-23	2021-12-23	PEP 494
3.5	security	2015-09-13	2020-09-13	PEP 478
2.7	end-of-life	2019-07-03	2020-01-01	PEP 373

그리고 반드시 설치를 진행할 때 PATH로 python을 잡는 것이 환경구성에 유리하다

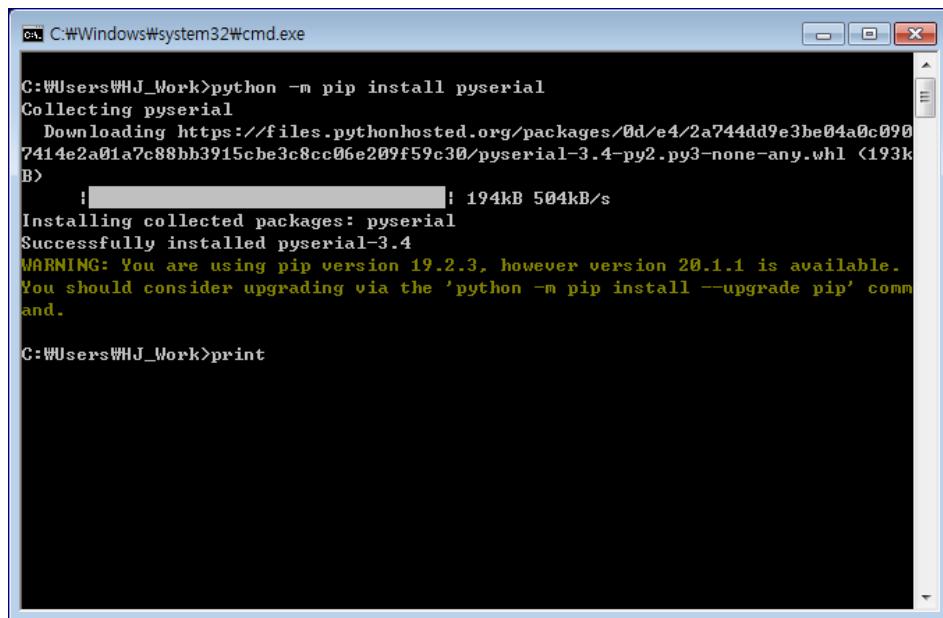


- B. 파이썬 환경을 구성했다면, 시리얼 통신에서 사용될 pyserial 패키지를 다운로드 하려고 한다.
원도우 cmd창을 열도록 한다. 먼저 파이썬이 먼저 설치되어 있는지 확인한다



```
C:\Windows\system32\cmd.exe - python
C:\Users\HJ_Work>python
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

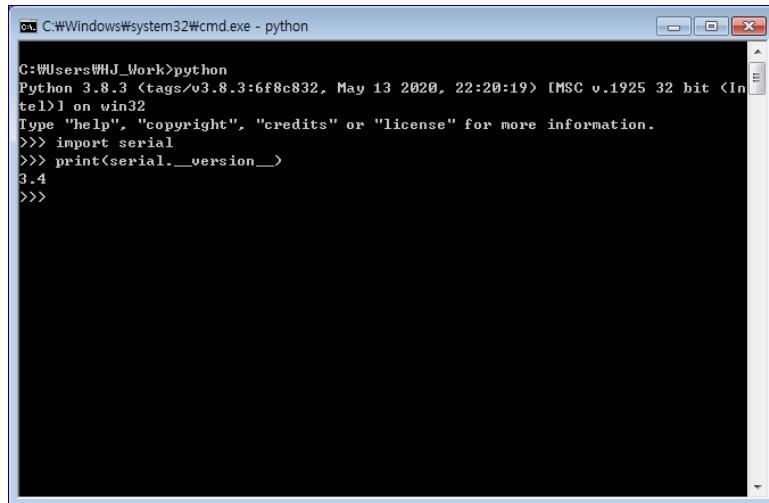
> python -m pip install pyserial



```
C:\Windows\system32\cmd.exe
C:\Users\HJ_Work>python -m pip install pyserial
Collecting pyserial
  Downloading https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c0907414e2a01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl (193kB)
    ! [██████████] 194kB 504kB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.4
WARNING: You are using pip version 19.2.3, however version 20.1.1 is available.
You should consider upgrading via the 'python -m pip install --upgrade pip' command.

C:\Users\HJ_Work>print
```

설치가 정상적으로 되었는지, 버전을 확인하도록 한다.



```
C:\Windows\system32\cmd.exe - python
C:\Users\WJ_Work>python
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import serial
>>> print(serial.__version__)
3.4
>>>
```

4.2 main 작성

A. COM port 확인

원도우에서의 시리얼 포트는 COMx로 표현된다

```
# 시리얼 COMPORT 입력
name = input("Enter the Port Name of your device (Ex: COM3):")
ret = 0
# COMPORT 확인
ret=Serial_Port_Configuration(name)
# 시스템 종료
if(ret < 0):
    decode_menu_command_code(0)
```

시리얼 포트 사용법은 주석된 링크에서 확인할 수 있다. 만일 포트를 열 수 없다면 시스템 상에서 사용이 가능한 시리얼 포트를 출력해준다

```
def Serial_Port_Configuration(port):
    # API 사용법 https://pyserial.readthedocs.io/en/latest/shortintro.html
    # window, linux 모두 사용 가능
    global ser
    try:
        ser = serial.Serial(port,115200,timeout=2)
    except:
        print("\n    Oops! That was not a valid port")

    # 시스템 사용 가능한 port 번호를 출력
    port = serial_ports()
    if(not port):
        print("\n    No ports Detected")
    else:
        print("\n    Here are some available ports on your PC. Try Again!")
        print("\n        ",port)
    return -1
if ser.is_open:
    print("\n    Port Open Success")
else:
    print("\n    Port Open Failed")
return 0
```

`sys.platform.startswith`를 통해서 현재 OS를 구분한다. 그리고 그에 맞는 시리얼 포트 형식을 구분해서 ports에 리스트 형태로 담는다. 그리고 'serial.Serial' API로 열리는 COM(윈도우 기준) 포트만 반환을 한다

```
def serial_ports():
    """ Lists serial port names
        :raises EnvironmentError:
            On unsupported or unknown platforms
        :returns:
            A list of the serial ports available on the system
    """
    if sys.platform.startswith('win'):
        ports = ['COM%d' % (i + 1) for i in range(256)]
    elif sys.platform.startswith('linux') or sys.platform.startswith('cygwin'):
        # this excludes your current terminal "/dev/tty"
        # glob 모듈의 glob 함수는 사용자가 제시한 조건에 맞는 파일명을 리스트 형식으로 반환한다
        ports = glob.glob('/dev/tty[A-Za-z]*')
    elif sys.platform.startswith('darwin'):
        ports = glob.glob('/dev/tty.*')
    else:
        raise EnvironmentError('Unsupported platform')

    result = []
    for port in ports:
        try:
            s = serial.Serial(port)
            s.close()
            result.append(port)
        except (OSError, serial.SerialException):
            pass
    return result
```

최종적으로 `Serial_Port_Configuration`을 통해서 나온 반환값에 따라 아래 명령을 진행하게 된다. Python은 기본적으로 string 입력이다. 따라서 입력에 대해서 `isdigit` 함수를 통해 숫자로 된 문자인지 검사하게 된다. 검사가 `true`를 반환하면 해당 명령에 대한 `decode`를 실행한다

```
while True:
    print("\n +=====\n")
    print(" |       Menu           |")
    print(" |       STM32F4 BootLoader v1 |")
    print(" +=====+\n")

    print("\n Which BL command do you want to send ??\n")
    print("   BL_GET_VER           --> 1")
    print("   BL_GET_HLP            --> 2")
    print("   BL_GET_CID             --> 3")
    print("   BL_GET_RDP_STATUS      --> 4")
    print("   BL_GO_TO_ADDR          --> 5")
    print("   BL_FLASH_MASS_ERASE     --> 6")
    print("   BL_FLASH_ERASE          --> 7")
    print("   BL_MEM_WRITE            --> 8")
    print("   BL_EN_R_W_PROTECT        --> 9")
    print("   BL_MEM_READ             --> 10")
    print("   BL_READ_SECTOR_P_STATUS    --> 11")
    print("   BL_OTP_READ              --> 12")
    print("   BL_DIS_R_W_PROTECT        --> 13")
    print("   BL_MY_NEW_COMMAND         --> 14")
    print("   MENU_EXIT                  --> 0")

    command_code = input("\n   Type the command code here :")

    # 입력값이 숫자인지 검사
    if(not command_code.isdigit()):
        print("\n   Please Input valid code shown above")
    else:
        decode_menu_command_code(int(command_code))

    input("\n   Press any key to continue  :")
    # 버퍼 flush
    purge_serial_port()
```

4.3 decode command

- 사용자가 입력한 명령에 대해서 나눠서 처리하는 부분이다. 먼저 0일 때는 잘못 기입한 입력이 들어왔을 때로, 예외로 처리하고 시스템을 종료하게 된다

A. 공통 함수

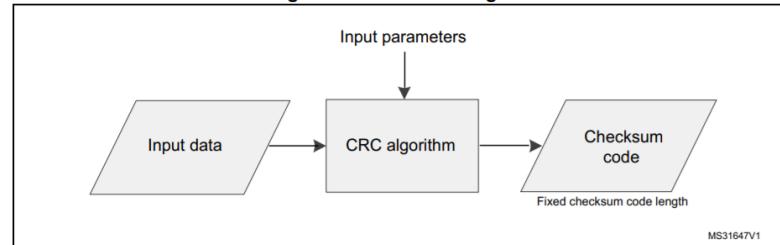
a. get_crc

```
def get_crc(buff, length):
    Crc = 0xFFFFFFFF
    #print(length)
    for data in buff[0:length]:
        Crc = Crc ^ data
        for i in range(32):
            if(Crc & 0x80000000):
                Crc = (Crc << 1) ^ 0x04C11DB7
            else:
                Crc = (Crc << 1)
    return Crc
```

명령 바이트에 대한 crc 4바이트를 만들게 된다. 그런데 STM32에서 사용하는 crc 알고리즘을 사용할 것이기 때문에 HOST에서도 반드시 해당 알고리즘을 사용해야만 서로 일치하는 값을 가질 수 있다

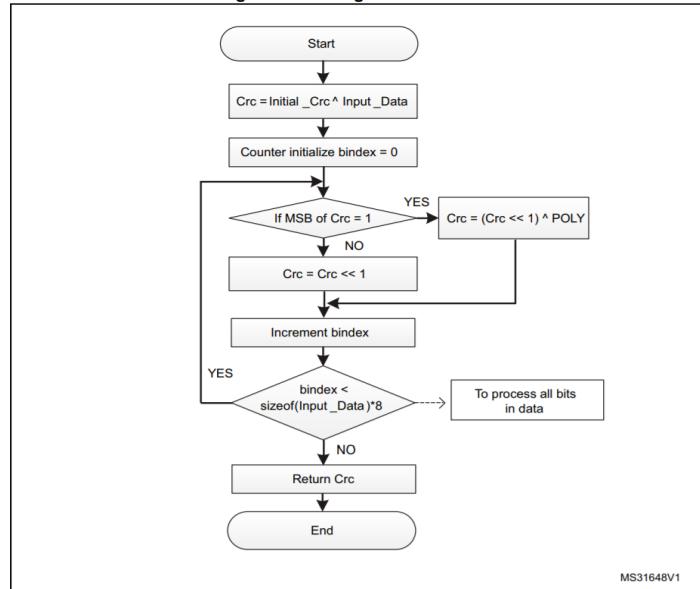
다시 한 번 STM32 CRC 알고리즘을 확인해보도록 한다

Figure 1. CRC block diagram



MS31647V1

Figure 2. CRC algorithm flowchart



MS31648V1

데이터시트에 나와있는 알고리즘 계산 예시다. 최상위 비트가 1이라면 POLY와 xor 연산을 하게 된다

Figure 3. Step-by-step CRC computing example

bindex	Execution step	Binary format	Hex
	Crc = Initial_Crc ^ Input_Data	1 1 1 1 1 1 1 1 (Initial_Crc) ^ 1 1 0 0 0 0 0 1 (Input_Data)	0xFF
0	Crc << 1	<u>0 0 1 1 1 1 1 0</u>	0xC1
1	Crc << 1	<u>0 1 1 1 1 1 0 0</u>	0x3E
2	Crc << 1	<u>1 1 1 1 1 0 0 0</u>	0x7C
	Crc = Crc ^ POLY	<u>1 1 1 1 0 0 0 0</u> ^ 1 1 0 0 1 0 1 1 (POLY)	0xF8
3	Crc << 1	<u>0 0 1 1 1 0 1 1</u>	0xFO
4	Crc << 1	<u>0 1 1 1 0 1 1 0</u>	0xCB
5	Crc << 1	<u>1 1 0 1 0 1 1 0</u>	0x3B
	Crc = Crc ^ POLY	<u>1 1 0 1 1 0 0 0</u> ^ 1 1 0 0 1 0 1 1 (POLY)	0x76
6	Crc << 1	<u>0 0 0 1 0 0 1 1</u>	0xEC
7	Crc << 1	<u>0 0 1 0 0 1 1 0</u>	0xD8
	Crc (Returned value)	0 1 0 0 1 1 0 0	0xCB
			0x13
			0x26
			0x4C

그리고 POLY는 STM32F429에서는 0x04C11DB7로 정의되어 있다

3 CRC migration through STM32 series

The CRC peripheral features can vary from one STM32 series to another. [Table 5](#) lists the CRC features and offers a software compatibility analysis for each STM32 series.

Table 5. STM32 CRC peripheral features

Feature	F1 series	L1 series	F2 series	F4 series	F0 series	F3 series
Single input/output 32-bit data register				YES		
General-purpose 8-bit register				YES		
Input buffer to avoid bus stall during calculation			NO		YES	
Reversibility option on I/O data			NO		YES	
CRC initial value			Fixed to 0xFFFFFFFF	Programmable on 32 bits	Programmable on 8, 16, 32 bits	
Handled data size in bits			32		8, 16, 32	
Polynomial size in bits			32		7, 8, 16, 32	
Polynomial coefficients			Fixed to 0x4C11DB7		Programmable	

따라서 HOST에서 python으로 다음과 같이 32비트만큼 루프를 도는 코드를 작성할 수 있다

b. Write_to_serial_port

```
def Write_to_serial_port(value, *length):
    data = struct.pack('>B', value)
    if (verbose_mode):
        value = bytearray(data)
        #print("    "+hex(value[0]), end='')
        print("    "+"0x{:02x}".format(value[0]),end=' ')
    if(mem_write_active and (not verbose_mode)):
        print("#",end=' ')
    ser.write(data)
```

struct.pack은 파이썬에서 C프로그램으로 데이터를 전달할 때, C에 적합한 형식으로 보내기 위해서 사용된다. C에 적합한 형식은 파이썬에서 정의되어 있다

Format	C Type	Python type	Standard size	Notes
x	pad byte	no value		
c	char	bytes of length 1	1	
b	signed char	integer	1	(1),(3)
B	unsigned char	integer	1	(3)
?_Bool	bool	bool	1	(1)
h	short	integer	2	(3)
H	unsigned short	integer	2	(3)
i	int	integer	4	(3)
I	unsigned int	integer	4	(3)
l	long	integer	4	(3)
L	unsigned long	integer	4	(3)
q	long long	integer	8	(2), (3)
Q	unsigned long long	integer	8	(2), (3)
n	ssize_t	integer		(4)
N	size_t	integer		(4)
e	(7)	float	2	(5)
f	float	float	4	(5)
d	double	float	8	(5)
s	char[]	bytes		
p	char[]	bytes		
P	void *	integer		(6)

그리고 >은 빅 엔디안 형식으로 파일 내용을 저장하겠다는 의미다. 아래 예시를 보면 실제로 바이트 순서에 따라서 사이즈가 달라지는 것을 볼 수 있다. 추측하건대 빅 엔디안(>) 형식으로 쓰면 그대로 h(2)i(4)가 들어가므로 6이지만, default로는 i(4)h(2) 그런데 h쪽에 2바이트가 붙어서 구분짓기(?) 때문에 8바이트로 되는 것 같다. 현재는 한 바이트라서 크게 상관은 없을 것이다

```
>>> from struct import *
>>> calcsize('hi')
8
>>> calcsize('>hi')
6
>>>
```

[관련 내용 출처](#)

c. read_bootloader_reply

host에서 사용자 명령을 보내고 난 후 target에서 이를 처리하고 난 후 처리 결과에 대한 ACK/NACK를 전달하게 된다. target에서 정의한 커맨드 포맷은 (ACK/답 length), (NACK)과 같다. 따라서 ACK reply는 총 2바이트, NACK reply는 총 1바이트 답이 오게 된다.

```
def read_bootloader_reply(command_code):
    len_to_follow=0
    ret = -2

# ack은 길이가 2인 리스트
    ack=read_serial_port(2)
    if(len(ack)):
        a_array=bytarray(ack)
        if (a_array[0]== 0xA5):
            len_to_follow=a_array[1]
            print("\n  CRC : SUCCESS Len : ",len_to_follow)
            if (command_code) == COMMAND_BL_GET_VER :
                process_COMMAND_BL_GET_VER(len_to_follow)

            elif(command_code) == COMMAND_BL_GET_HELP:
                process_COMMAND_BL_GET_HELP(len_to_follow)

            elif(command_code) == COMMAND_BL_GET_CID:
                process_COMMAND_BL_GET_CID(len_to_follow)

            elif(command_code) == COMMAND_BL_GET_RDP_STATUS:
                process_COMMAND_BL_GET_RDP_STATUS(len_to_follow)

            elif(command_code) == COMMAND_BL_GO_TO_ADDR:
                process_COMMAND_BL_GO_TO_ADDR(len_to_follow)

            elif(command_code) == COMMAND_BL_FLASH_ERASE:
                process_COMMAND_BL_FLASH_ERASE(len_to_follow)

            elif(command_code) == COMMAND_BL_MEM_WRITE:
                process_COMMAND_BL_MEM_WRITE(len_to_follow)

            elif(command_code) == COMMAND_BL_READ_SECTOR_P_STATUS:
                process_COMMAND_BL_READ_SECTOR_STATUS(len_to_follow)

            elif(command_code) == COMMAND_BL_EN_R_W_PROTECT:
                process_COMMAND_BL_EN_R_W_PROTECT(len_to_follow)

            elif(command_code) == COMMAND_BL_DIS_R_W_PROTECT:
                process_COMMAND_BL_DIS_R_W_PROTECT(len_to_follow)
```

그래서 read_serial_port 함수를 통해서 길이가 2인 리스트를 전달받는다. 참고로 bytarray 1바이트 단위의 값을 연속적으로 저장하는 시퀀스 자료형이다. 다른 자료형 bytes와 차이점은 요소를 변경할 수 있느냐의 차이가 있다. bytes는 요소를 변경할 수 없고, bytarray는 요소를 선언 이후에도 변경할 수 있다

그래서 첫 데이터가 0xA5일 때와 0x7F일 때를 나눠서 처리하게 된다

B. BL_GET_VER

a. 코드

```
def decode_menu_command_code(command):
    ret_value = 0
    data_buf = []
    for i in range(255):
        data_buf.append(0)

    if(command == 0):
        print("\n    Exiting...!")
        raise SystemExit

    # 명령에 따라서 달라진다

    elif(command == 1):
        print("\n    Command == > BL_GET_VER")
        COMMAND_BL_GET_VER_LEN = 6
        data_buf[0] = COMMAND_BL_GET_VER_LEN-1
        data_buf[1] = COMMAND_BL_GET_VER
        crc32 = get_crc(data_buf,COMMAND_BL_GET_VER_LEN-4)
        crc32 = crc32 & 0xffffffff
        data_buf[2] = word_to_byte(crc32,1,1)
        data_buf[3] = word_to_byte(crc32,2,1)
        data_buf[4] = word_to_byte(crc32,3,1) |
        data_buf[5] = word_to_byte(crc32,4,1)

        Write_to_serial_port(data_buf[0],1)
        for i in data_buf[1:COMMAND_BL_GET_VER_LEN]:
            Write_to_serial_port(i,COMMAND_BL_GET_VER_LEN-1)

        ret_value = read_bootloader_reply(data_buf[1])
```

b. BL_GET_VER 명령 구조

맨 앞에 길이 정보와 그 뒤에 부트로더 버전 명령 번호와 마지막으로 데이터 무결성 확인을 위한 CRC값이 담아 있다

길이	명령 번호	4바이트 CRC
----	-------	----------

c. process_COMMAND_BL_GET_VER

Target으로부터 부트로더 버전에 대한 답을 받게 된다

```
def process_COMMAND_BL_GET_VER(length):
    ver=read_serial_port(1)
    value = bytearray(ver)
    print("\n    Bootloader Ver. : ",hex(value[0]))
```

C. BL_GET_HELP

a. 코드

```
elif(command == 2):
    print("\n    Command == > BL_GET_HELP")
    COMMAND_BL_GET_HELP_LEN = 6
    data_buf[0] = COMMAND_BL_GET_HELP_LEN-1
    data_buf[1] = COMMAND_BL_GET_HELP
    crc32 = get_crc(data_buf,COMMAND_BL_GET_HELP_LEN-4)
    crc32 = crc32 & 0xffffffff
    data_buf[2] = word_to_byte(crc32,1,1)
    data_buf[3] = word_to_byte(crc32,2,1)
    data_buf[4] = word_to_byte(crc32,3,1)
    data_buf[5] = word_to_byte(crc32,4,1)

    Write_to_serial_port(data_buf[0],1)
    for i in data_buf[1:COMMAND_BL_GET_HELP_LEN]:
        Write_to_serial_port(i,COMMAND_BL_GET_HELP_LEN-1)
```

b. BL_GET_HELP 명령 구조

BL_GET_VER과 유사하다. 길이 정보와 HELP 명령 번호, 그리고 무결성을 위한 CRC 가 담아져서 총 6바이트가 전송된다

길이	명령 번호	4바이트 CRC
----	-------	----------

c. process_COMMAND_BL_GET_HELP

Target으로 받은 길이만큼 HELP 정보를 받게 된다. VER과 다르게 여러 바이트에 해당하는 데이터이므로 상황에 따라 길이가 다르게 된다

```
def process_COMMAND_BL_GET_HELP(length):
    #print("reading:", length)
    value = read_serial_port(length)
    reply = bytearray(value)
    print("\n    Supported Commands : ", end=' ')
    for x in reply:
        print(hex(x), end=' ')
    print()
```

단순하게 hex 값만 출력하는데, 초기 화면 데이터 값으로 무슨 명령인지 확인해야 하는 불편함이 있긴하다

5. USB DFU 부트로더 작성

- STM에서 제공했던 내장 부트로더는 UART 통신만 지원했다. 쉽게 펌웨어 업데이트가 가능했지만, USB 인터페이스가 지원되지 않았다. 양산, 유지보수 과정을 생각해보면 USB 인터페이스의 필요성은 없지 않다

5.1 USB 기본 개념

A. USB Device

- A Host(MCU) 요청에 의해 동작
- B 각 디바이스는 디바이스 주소를 가지고 있어, 디바이스 주소가 같으면 수신 버퍼에 저장하고 인터럽트를 발생한다. 다르면 무시하게 된다
- C Host의 표준 request에 응답. 처음 디바이스 열거를 위해 host가 보내는 request로 기능과 상태를 묻는다
- D 에러 확인
- E 전원관리. 버스 활동이 없으면 전류 사용 제한
- F Host 동작에 의존하는 형태로, 데이터 교환에 있어 host에 응답을 보낸다

5.2 USB Key를 이용한 펌웨어 업데이트

- STM에서 제공하는 ‘Upgrading STM32F4DISCOVERY board firmware using a USB key’ 제목의 AN3990 Application note를 참조해서 예제 프로그램을 실행해보려고 한다. 차례대로 문서를 분석해보려고 한다
- 문서는 다음 링크(https://www.st.com/resource/en/application_note/dm00039672-upgrading-stm32f4discovery-board-firmware-using-a-usb-key-stmicroelectronics.pdf)를 참조한다

A. Introduction

Introduction

An important requirement for most Flash memory-based systems is the ability to update the firmware installed in the end product. This document provides general guidelines for creating a firmware upgrade application based on the STM32F4DISCOVERY board.

The STM32F4 series microcontroller can run user-specific applications to upgrade the firmware of the microcontroller-embedded Flash memory. This feature allows the use of any type of communication protocol for the reprogramming process (for example, CAN, USART and USB). USB Host mass storage is the example used in this application note.

The firmware upgrade using a USB Host is very advantageous because it is a standalone executed code in which the user does not need to use a host computer to perform the firmware upgrade. The user only needs a Flash disk to upgrade the target STM32 device.

Document contents

- *Section 1: Firmware upgrade overview* contains an overview of the firmware upgrade process and demonstrates how to run the firmware upgrade.
- *Section 2: How to use the firmware upgrade application* describes the user program and system requirements for the software and hardware.

Reference documents

- STM32F4DISCOVERY STM32F4 high-performance discovery board (UM1472)
- STM32F405xx, STM32F407xx, STM32F415xx and STM32F417xx advanced ARM-based 32-bit MCUs reference manual (RM0090)
- STM32F405xx STM32F407xx datasheet
- STM32F415xx STM32F417xx datasheet

The above documents are available at www.st.com/stm32f4-discovery.

- a. usb key를 이용한 firmware update와 관련된 문서다
- b. overview와 사용법에 대해서 문서는 구성되었다

1 Firmware upgrade overview

To program the firmware upgrade application to the Flash memory, use the STM32F4xx's embedded Bootloader or any in-system programming tool to easily reprogram this application.

The firmware upgrade application uses the USB Host to:

- Download a binary file (.bin) from a Flash disk (thumb drive) to the STM32F4xx's internal flash memory.
- Upload all the STM32F4xx's internal Flash memory content into a binary file.
- Execute the user program.

Note: This application note is based on the STM32 USB On-The-Go (OTG) Host and device library. For more details about the USB Host stack and a mass storage demonstration, please refer to user manual (UM1021).

1.1 Implementing the firmware upgrade application

The firmware upgrade application contains the source files in [Table 1](#).

Table 1. Source files

File	Contents
main.c	Contains the USB initialization data. The USB Host state machine is then executed if the user wants to execute the firmware upgrade application or the program will execute the user code
stm32f4xx_it.c	Contains the interrupt handlers for the application
command.c	Contains the firmware upgrade commands (DOWNLOAD, UPLOAD and JUMP commands)
flash_if.c	Provides a medium layer access to the STM32 embedded Flash driver
usb_bsp.c	Implements the board support package for the USB Host library
usbh_usr.c	Includes the USB Host library user callbacks
system_stm32f4xx.c	Contains the system clock configuration for STM32 F4xx devices

- a. usb host를 사용하여 Flash disk (thumb drive)에서 STM32F4xx의 내부 Flash 메모리로 bin 파일을 다운로드 하게 된다
- b. 예제 프로그램은 다음과 같은 소스파일들로 구성되어 있다
- c. main.c: usb 초기화, 사용자가 원할 시에 firmware upgrade 프로그램이 실행된다. 그렇지 않으면 user code가 실행된다
- d. stm32f4xx_it.c: 인터럽트 핸들러
- e. command.c: firmware upgrade 명령 정의
- f. flash_if.c: stm32 flash driver에 접근하기 위한 medium layer
- g. usb_bsp.c: usb host library를 위한 BSP
- h. usbh_usr.c: usb host library user callback
- i. system_stm32f4xx.c: rcc clock configuration이 담겨 있다

After the board reset and depending on the user button state:

1. **User button pressed:** The firmware upgrade application is executed.
2. **User button not pressed:** A test on the user application start address will be performed and one of the below processes is executed.
 - User vector table available: User application is executed.
 - User vector table not available: firmware upgrade application is executed.

During the firmware upgrade application execution, there is a continuous check on the user button pressed state time. Depending on the state time of the user button, one of the following processes is executed.

Table 2. User button state time control

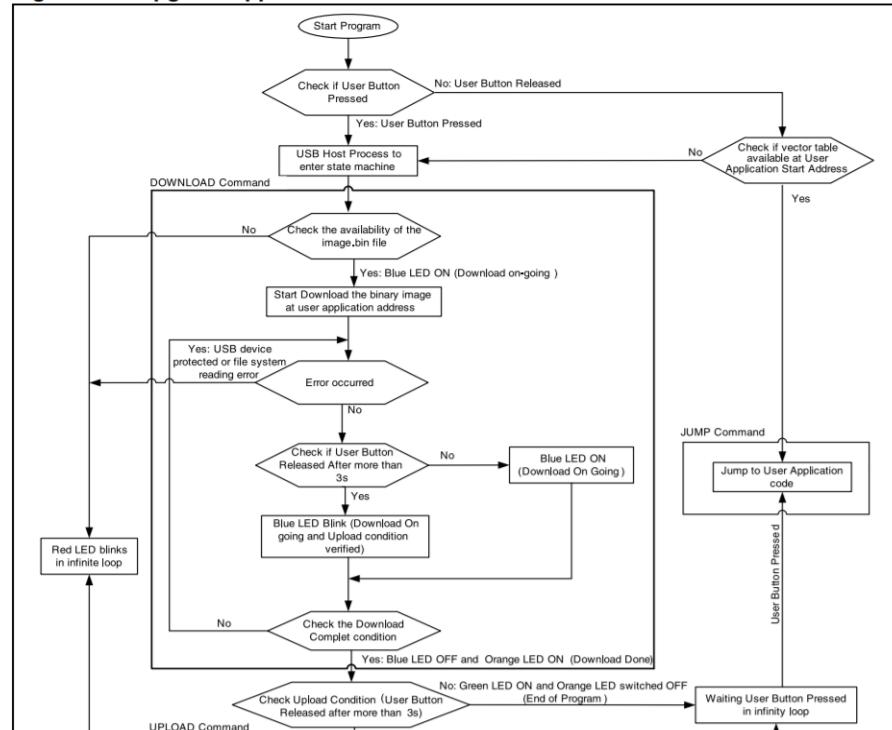
User button state	Time	Process executed
Pressed	> 3 seconds	UPLOAD command will be executed immediately after completed execution of the DOWNLOAD command.
	< 3 seconds	Only the DOWNLOAD command is executed.

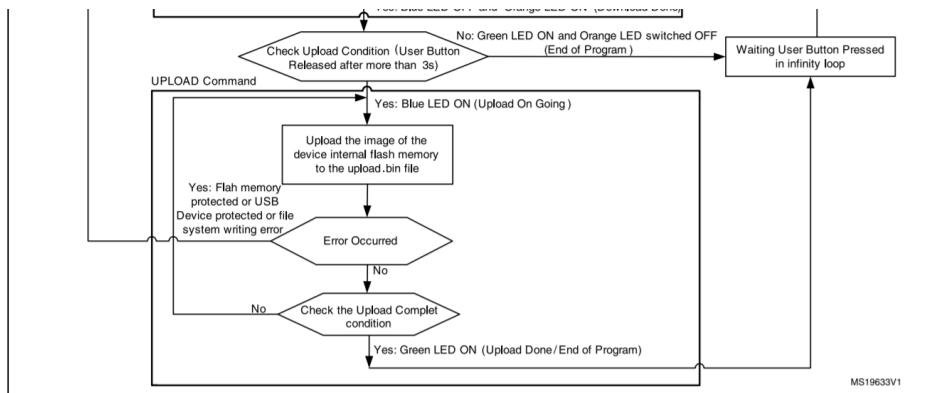
Note: The UPLOAD command condition verification is signaled by the blinking state of the blue LED.

- j. user button이 눌렸을 시 firmware upgrade가 실행된다
- k. 눌리지 않았다면 프로그램이 써져 있는지 검사를 하게 된다. 예제 프로그램에서는 MSP값을 검사하게 된다
- l. 그리고 버튼이 눌렸다면, 눌린 시간을 체크하게 된다. 3초 이상 눌린다면 다운로드 후 업로드까지 한다. 그 이하이면 다운로드만 하게 된다

C. Flowchart

Figure 1. Upgrade application flowchart

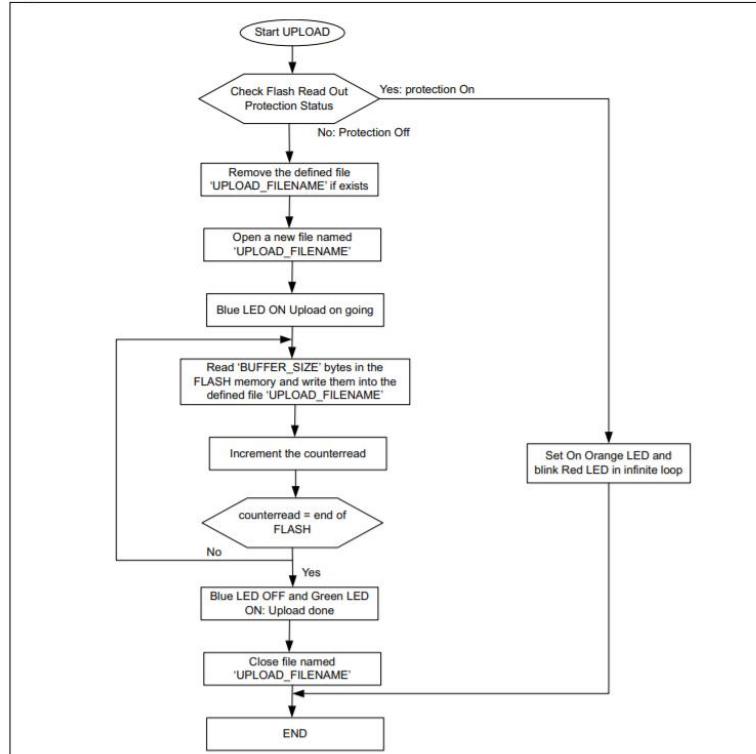




- 프로그램이 시작된다
- User 버튼이 눌렸는지 확인한다. 안눌렸으면 user program을 실행한다
- USB 초기화
- image.bin 파일이 있는지 확인한다. 있다면 blue(green) led가 켜질 것이다
- 그리고 3초 동안 delay를 가지면서 버튼이 그 이후에도 눌렸는지 확인한다
- 먼저 3초 이후 버튼이 눌린 여부는 상관하지 않고, 초기에 버튼이 눌렸다면 펌웨어 업그레이드를 진행한다
- 만일 3초 이후에도 버튼이 눌렸다면 업로드를 실행하도록 한다
- 그리고 User 어플리케이션으로 jump해서 실행하도록 한다

D. Upload Flowchart

Figure 3. UPLOAD command



- a. 파일 이름이 존재한다면 먼저 삭제하도록 한다
- b. Flash 영역 끝까지 읽으면서 파일에 쓰도록 한다
- c. 만일 Flash 메모리가 protection이 켜있는 상태라면 위 과정이 불가능하다

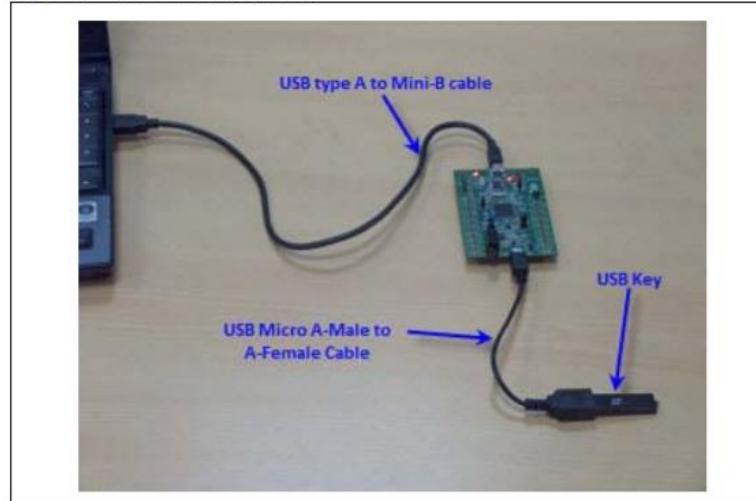
E. 시스템 요구사항

2 How to use the firmware upgrade application

2.1 System requirements

Before running your application, you should establish the connection with the STM32F4DISCOVERY board as following in [Figure 5](#).

Figure 5. Hardware environment



To run the firmware upgrade application on your STM32F4DISCOVERY board, the minimum requirements are as follows:

- Microsoft® Windows PC (2000, XP, Vista, 7)
- USB type A to Mini-B' cable, used to power the board (through USB connector CN1) from host PC and connect to the embedded ST-LINK/V2 for debugging and programming.
- USB micro A-Male to A-Female' cable, used to connect the USB key (through USB connector CN5) as USB Device to host STM32F4xx.

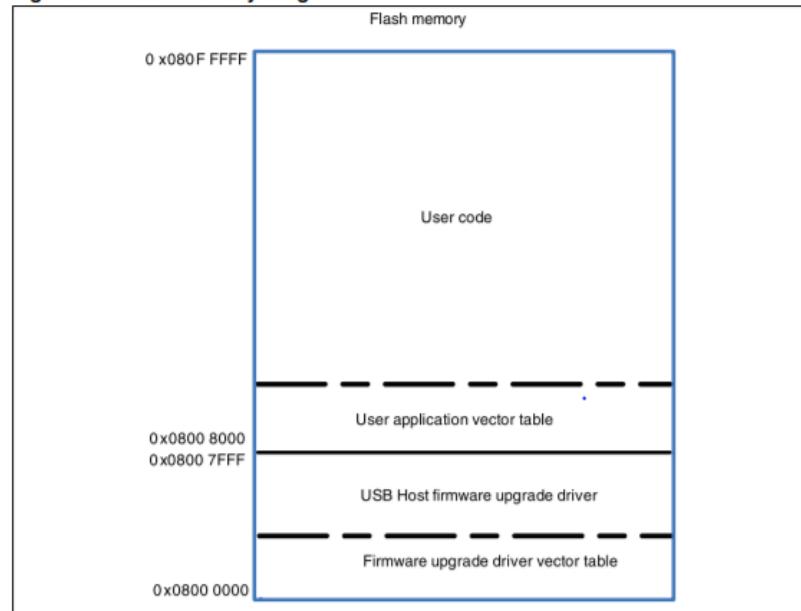
- a. 먼저 보드와 프로그램 다운을 위한 케이블이 있어야 한다
- b. 그리고 USB micro A-Male to A-Female Cable을 따로 준비해야 한다
- c. 펌웨어 업그레이드를 하기 위해서는 image.bin 파일과 담은 USB key를 가지고온다
- d. image.bin 파일은 반드시 아래 요구사항을 따라야 한다

2.3 User program condition

The user application (binary file) to be loaded into the Flash memory using the firmware upgrade application is built by the following configuration settings:

1. Set the program load address to APPLICATION_ADDRESS in the toolchain linker file.
2. Relocate the vector table to address APPLICATION_ADDRESS using the NVIC_SetVectorTable function or the VECT_TAB_OFFSET definition inside the system_stm32f4xx.c file.

Figure 6. Flash memory usage



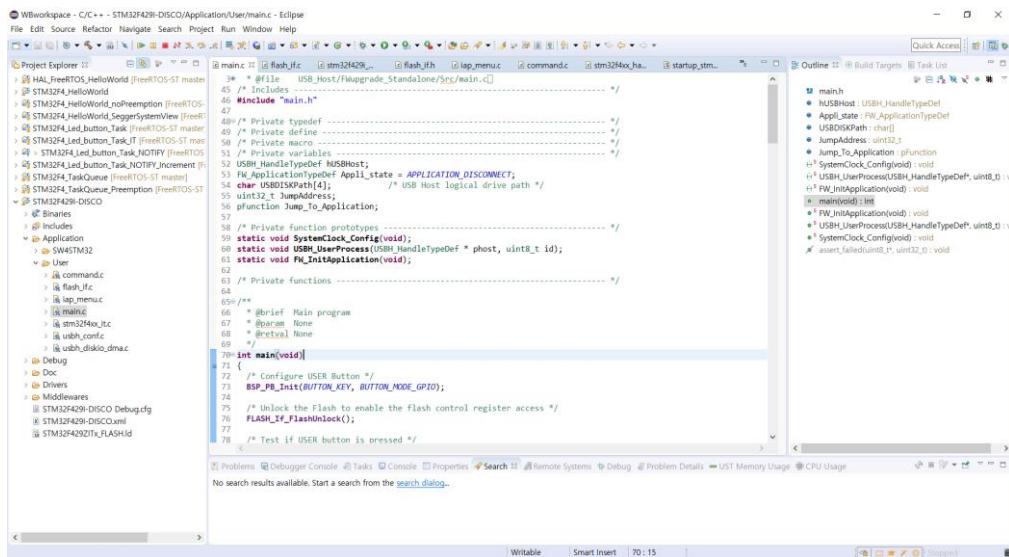
시작 address가 APPLICATION_ADDRESS와 같아야 한다. Program할 때 Linker에서 설정할 수 있다. 그리고 VECT_TAB_OFFSET을 (APPLICATION_ADDRESS 0x08000000)으로 수정해야 한다.

F. 실습

위 문서 설명과 예제는 STM32F4-Discovery 기반이다. 따라서 이어지는 다음 문서 내용과 다르게 다를 것이다. 지금부터 다를 프로젝트는 '[STM32Cube_FW_F4_V1.25.0](#)' Cube 라이브러리를 참고할 것이다. 그러면 32F429ZI-DISC1에 맞는 USB Firmware 다운로드 프로젝트를 찾도록 한다. 해당 라이브러리에는 '[STM32Cube_FW_F4_V1.25.0/Projects/STM32F429I-Discovery/Applications/USB_Host/FWupgrade_Standalone](#)' 위치에 있으니 참고하도록 한다

G. 프로젝트 열기

프로젝트에 대해서 다양한 toolchain을 제공해주고 있다. 원하는 toolchain으로 실행하면 될 것 같다. 이번에는 'SW4STM32'을 활용하려고 한다.



이 프로젝트가 현재 32F429-DISC1에 맞지 않는다. 보통 버튼을 눌렀을 때만 펌웨어 업데이트가 진행되어야 하는데 그렇게 진행되지 않고 있다. 따라서 'BSP_PB_GetState(BUTTON_KEY) != GPIO_PIN_RESET' 부분에서 GPIO_PIN_RESET을 GPIO_PIN_SET으로 바꿔준다

참고 파일은 (src/workbench_workspace/01_FWupgrade_Standalone/)의 iap_menu.c, main.c를 담아 놓을 것이니 덮어쓰거나 웃 부분을 찾아서 수정하면 될 것이다

H. 소스코드 분석

a. Jump to user application (0x0800C000)

```
/* Test if USER button is pressed */
if (BSP_PB_GetState(BUTTON_KEY) != GPIO_PIN_SET)
{
    /* Check Vector Table: Test if user code is programmed starting from
     * address "APPLICATION_ADDRESS" */
    if ((((*_IO uint32_t *) APPLICATION_ADDRESS) & 0xFF000000) == 0x20000000)
        || (((*_IO uint32_t *) APPLICATION_ADDRESS) & 0xFF000000) ==
            0x10000000)
    {
        /* Jump to user application */
        JumpAddress = (*(_IO uint32_t *) (APPLICATION_ADDRESS + 4));
        Jump_To_Application = (pFunction) JumpAddress;
        /* Initialize user application's Stack Pointer */
        __set_MSP((*_IO uint32_t *) APPLICATION_ADDRESS);
        Jump_To_Application();
    }

    /* Define the address from where user application will be loaded.
     * Note: the 1st and the second sectors 0x08000000-0x0800BFFF are reserved
     * for the Firmware upgrade code */
    #define APPLICATION_ADDRESS      (uint32_t)0x0800C000
}
```

만 첫 Flash에 MSP 값이 써져 있는지 확인한다. 있다면 user program이 있는지 판단한다

b. USB 초기화

```
HAL_Init();

/* Configure the system clock to 168 MHz */
SystemClock_Config();

/* Init FW upgrade Application */
FW_InitApplication();

/* Init Host Library */
USBH_Init(&hUSBHost, USBH_UserProcess, 0);

/* Add Supported Class */
USBH_RegisterClass(&hUSBHost, USBH_MSC_CLASS);

/* Start Host Process */
USBH_Start(&hUSBHost);
```

```
/* Run Application (Blocking mode) */
```

```
while (1)
```

```
{
```

```
    /* USB Host Background task */
```

```
    USBH_Process(&hUSBHost);
```

```
    /* FW Menu Process */
```

```
    FW_UPGRADE_Process();
```

```
}
```

usb 사용을 위한 clock, led 표시, host 라이브러리와 class 를 초기화한다. 그리고 usb core 상태 체크를 하게 된다. 마지막으로 'FW_UPGRADE_Process'에 진입하게 된다

c. FW_UPGRADE_Process

```
case DEMO_IAP:
    while (USBH_MSC_IsReady(&hUSBHost))
    {
        /* Control BUFFER_SIZE value */
        USBH_USR_BufferSizeControl();

        /* Keep LED1 and LED3 Off when Device connected */
        BSP_LED_Off(LED3);
        BSP_LED_Off(LED4);

        /* USER Button pressed Delay */
        IAP_UploadTimeout();

        /* Writes Flash memory */
        COMMAND_Download();

        /* Check if USER Button is already pressed */
        if ((UploadCondition == 0x01))
        {
            /* Reads all flash memory */
            COMMAND_Upload();
        }
        else
        {
            /* Turn LED4 Off: Download Done */
            BSP_LED_Off(LED4);
            /* Turn LED3 On: Waiting KEY button pressed */
            BSP_LED_On(LED3);
        }
    }

    /* Waiting USER Button Released */
    while ((BSP_PB_GetState(BUTTON_KEY) == GPIO_PIN_SET) &&
           (Appli_state == APPLICATION_READY))
    {
    }
```

IAP_UploadTimeout 함수에서 3 초 동안 delay 를 가진 다음 3 초 이상 늘렸다면, flash down 과 upload 모두 진행하고, 그렇지 않다면 flash down 만 하게 된다. 그리고 버튼이 release 된 것을 확인한 후에 application 으로 jump 하게 된다

d. COMMAND_Download

```
void COMMAND_Download(void)
{
    /* Open the binary file to be downloaded */
    if (f_open(&MyFileR, DOWNLOAD_FILENAME, FA_OPEN_EXISTING | FA_READ) == FR_OK)
    {
        if (f_size(&MyFileR) > USER_FLASH_SIZE)
        {
            /* No available Flash memory size for the binary file: Toggle LED4 in
             * infinite loop */
            Fail_Handler();
        }
        else
        {
            /* Download On Going: Turn LED4 On */
            BSP_LED_On(LED4);
            BSP_LED_Off(LED3);

            /* Erase FLASH sectors to download image */
            if (FLASH_If_EraseSectors(APPLICATION_ADDRESS) != 0x00)
            {
                /* Flash erase error: Toggle LED4 in infinite loop */
                BSP_LED_Off(LED4);
                Erase_Fail_Handler();
            }

            /* Program flash memory */
            COMMAND_ProgramFlashMemory();

            /* Download Done: Turn LED3 On and LED4 Off */
            BSP_LED_On(LED3);
            BSP_LED_Off(LED4);

            /* Close file */
            f_close(&MyFileR);
        }
    }
    else
    {
        /* If file does not exist, then turn LED4 On and LED3 Off */
        BSP_LED_On(LED4);
        BSP_LED_Off(LED3);
    }
}
```

쓰려고 하는 flash 영역을 지우고 난 후, program 을 실행한다

e. COMMAND_Upload

```

void COMMAND_Upload(void)
{
    __IO uint32_t address = APPLICATION_ADDRESS;
    __IO uint32_t counterread = 0x00;
    uint32_t tmpcounter = 0x00, indexoffset = 0x00;
    FlagStatus readoutstatus = SET;
    uint16_t byteswritten;

    /* Get the read out protection status */
    readoutstatus = FLASH_If_ReadOutProtectionStatus();
    if (readoutstatus == RESET)
    {
        /* Remove UPLOAD file if it exists on flash disk */
        f_unlink(UPLOAD_FILENAME);

        /* Init written byte counter */
        indexoffset = (APPLICATION_ADDRESS - USER_FLASH_STARTADDRESS);

        /* Open binary file to write on it */
        if ((Appli_state == APPLICATION_READY) &&
            (f_open(&MyFile, UPLOAD_FILENAME, FA_CREATE_ALWAYS | FA_WRITE) ==
             FR_OK))
        {
            /* Upload On Going: Turn LED4 On and LED3 Off */
            BSP_LED_On(LED4);
            BSP_LED_Off(LED3);

            /* Read flash memory */
            while ((indexoffset < USER_FLASH_SIZE) &&
                   (Appli_state == APPLICATION_READY))
            {
                for (counterread = 0; counterread < BUFFER_SIZE; counterread++)
                {
                    /* Check the read bytes versus the end of flash */
                    if (indexoffset + counterread < USER_FLASH_SIZE)
                    {
                        tmpcounter = counterread;
                        RAM_Buf[tmpcounter] = (*((uint8_t *) (address++)));
                }
            }
        }
    }
}

```

쓰고자 하는 파일을 open 한 후 flash 를 read 한 후 write 를 하는 것을 볼 수 있다
다운로드, 업로드 파일 이름은 이미 정의되어 있고, 수정이 가능하다

```

/* Private typedef ----- */
/* Private defines ----- */
#define UPLOAD_FILENAME          "0:UPLOAD.BIN"
#define DOWNLOAD_FILENAME         "0:image.BIN"

```

0:은 생략 가능한 부분이다

f. COMMAND_Jump

```

void COMMAND_Jump(void)
{
    /* Software reset */
    NVIC_SystemReset();
}

```

그리고 마지막에 SystemReset 으로 이 프로그램을 다시 시작하게 된다. System Control block 의 AIRCR 레지스터를 제어하면 pending 되는 것으로 추측된다

```

__NO_RETURN __STATIC_INLINE void __NVIC_SystemReset(void)
{
    __DSB();
    SCB->AIRCR = (uint32_t)((0x5FAUL << SCB_AIRCR_VECTKEY_Pos) |
                           (SCB->AIRCR & SCB_AIRCR_PRIGROUP_Msk) |
                           SCB_AIRCR_SYSRESETREQ_Msk); /* Ensure all outstanding memory accesses included
                                             buffered write are completed before reset */
    __DSB(); /* Keep priority group unchanged */
    __DSB(); /* Ensure completion of memory access */

    for(;;) /* wait until reset */
    {
        __NOP();
    }
}

typedef struct
{
    __IM uint32_t CPUID; /*!< Offset: 0x000 (R/_ ) CPUID Base Register */
    __IOM uint32_t ICSR; /*!< Offset: 0x004 (R/W) Interrupt Control and State Register */
    __IOM uint32_t VIOR; /*!< Offset: 0x008 (R/W) Vector Table Offset Register */
    __IOM uint32_t AIRCR; /*!< Offset: 0x00C (R/W) Application Interrupt and Reset Control Register */
    __IOM uint32_t SCCR; /*!< Offset: 0x010 (R/W) System Control Register */
    __IOM uint32_t CCR; /*!< Offset: 0x014 (R/W) Configuration Control Register */
    __IOM uint8_t SHP[12U]; /*!< Offset: 0x018 (R/W) System Handlers Priority Registers (4-7, 8-11, 12-15) */
    __IOM uint32_t SHCSR; /*!< Offset: 0x024 (R/W) System Handler Control and State Register */
    __IOM uint32_t CFSR; /*!< Offset: 0x028 (R/W) Configurable Fault Status Register */
    __IOM uint32_t HFSR; /*!< Offset: 0x02C (R/W) HardFault Status Register */
    __IOM uint32_t DFSR; /*!< Offset: 0x030 (R/W) Debug Fault Status Register */
    __IOM uint32_t MMFAR; /*!< Offset: 0x034 (R/W) MemManage Fault Address Register */
    __IOM uint32_t BFAR; /*!< Offset: 0x038 (R/W) BusFault Address Register */
    __IOM uint32_t AFSR; /*!< Offset: 0x03C (R/W) Auxiliary Fault Status Register */
    __IM uint32_t PFR[2U]; /*!< Offset: 0x040 (R/_ ) Processor Feature Register */
    __IM uint32_t DFR; /*!< Offset: 0x048 (R/_ ) Debug Feature Register */
    __IM uint32_t ADR; /*!< Offset: 0x04C (R/_ ) Auxiliary Feature Register */
    __IM uint32_t MMFR[4U]; /*!< Offset: 0x050 (R/_ ) Memory Model Feature Register */
    __IM uint32_t ISAR[5U]; /*!< Offset: 0x060 (R/_ ) Instruction Set Attributes Register */
    uint32_t RESERVED0[5U];
    __IOM uint32_t CPACR; /*!< Offset: 0x088 (R/W) Coprocessor Access Control Register */
} SCB_Type;

```

I. 실습



5.3 다른 방법으로 DFU 구현하기

- 이전 system rom bootloader에서 부트로더 코드가 이미 들어있었다고 했다. 따라서 JTAG이 없더라도 일반적인 펌웨어로 특정 flash에 다운로드할 수 있었다. 이전에는 uart 인터페이스를 사용했었고, 레퍼런스 문서를 좀 더 살펴보면 USB 인터페이스도 지원하는 것을 알 수 있었다

Embedded bootloader

The embedded bootloader mode is used to reprogram the Flash memory using one of the following serial interfaces:

- USART1 (PA9/PA10)
- USART3 (PB10/11 and PC10/11)
- CAN2 (PB5/13)
- **USB OTG FS (PA11/12) in Device mode (DFU: device firmware upgrade).**

The USART peripherals operate at the internal 16 MHz oscillator (HSI) frequency, while the CAN and USB OTG FS require an external clock (HSE) multiple of 1 MHz (ranging from 4 to 26 MHz).

The embedded bootloader code is located in system memory. It is programmed by ST during production. For additional information, refer to application note AN2606.

- 429 디스커버리 보드와 같은 경우에는 보드 회로 설계상 동작에 문제가 있는 것 같다. 실제로 DFU 소프트웨어를 킨 후, BOOT 핀 설정을 해도 동작을 하지 않는 것으로 보인다

출처는 다음과 같다

1. <https://community.st.com/s/question/0D50X00009XkgdfSAB/cannot-get-stm32f429-discovery-board-to-enter-dfu-mode>
2. https://embdev.net/articles/STM_Discovery_and_Nucleo_as_Black_Magic_Probe