

Custom STM32F429ZI 부트로더 만들기

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
2.	System Memory 부트로더	21
3.	Custom 부트로더 작성	22
4.	PC Host 통신 프로그램 작성 [Python]	23
5.	USB DFU 부트로더 작성.....	24
5.1	USB 기본 개념	24
5.2	USB Key를 이용한 펌웨어 업데이트	25

1. 부트로더 개요

- 부트로더는 단순하게 생각하면 작은 코드 덤머리라고 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

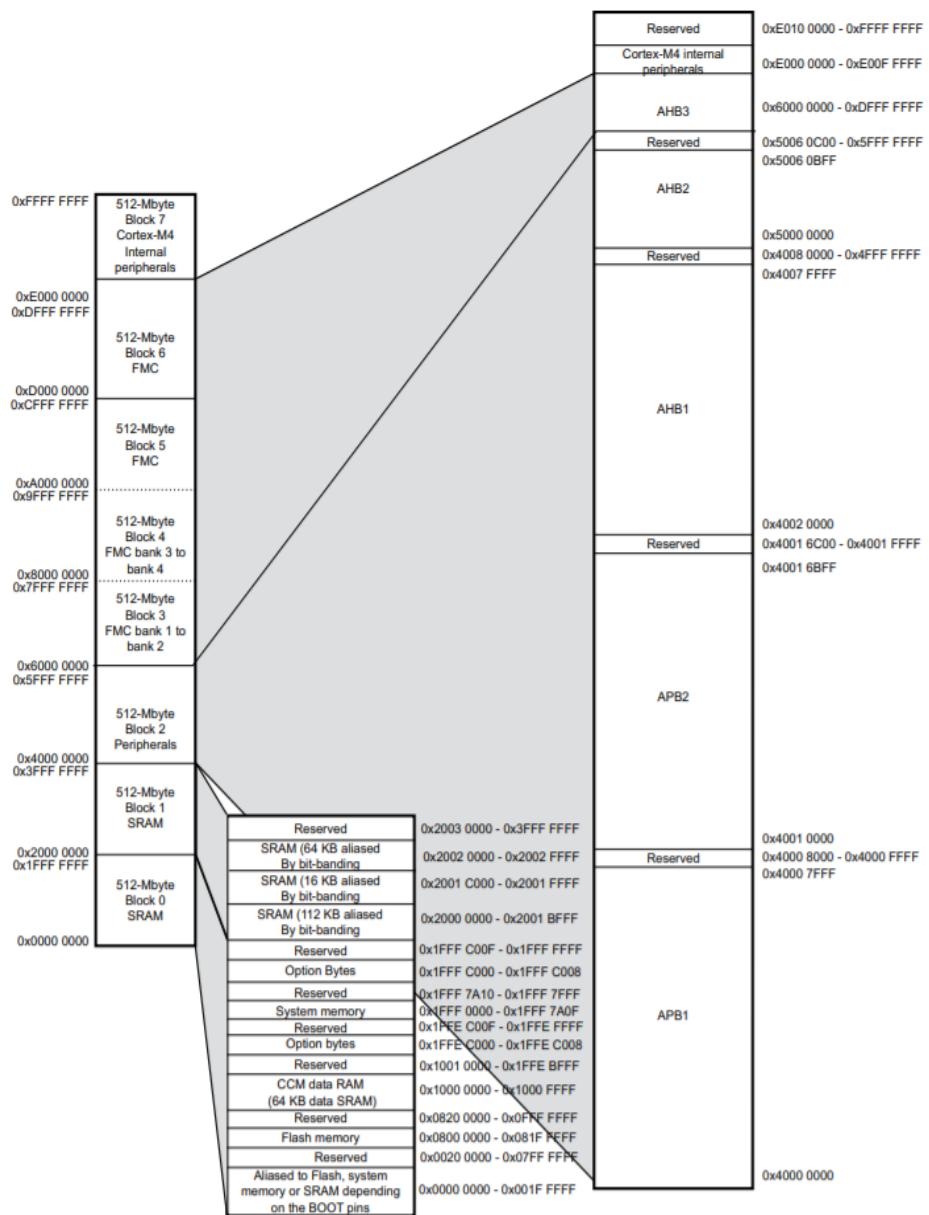
- a. 해당 보드는 부트로더를 가지고 있는가?
그렇다. Chip 안에 부트로더가 내장되어 있다
- b. MCU가 reset 때마다 동작을 하는가?
아니다. 기본적으로는 동작하지 않고, Boot Pin의 상태를 바꿔야만 Boot 코드가 동작한다
- c. 해당 보드에서의 부트로더의 주 역할은?
주로 바이너리를 다운로드, 업데이트를 할 때 사용된다
- d. 아두이노와 다르게 이 보드는 Bootloader가 매번 필요가 없는 이유는 ISP 장비로 ST-Link를 제공하고 있기 때문이다. 이는 해당 보드에 국한된 내용이 아닌 TI, 여타 보드 (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 꿰고 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. A 'Learning Platform' button is also present. Below the navigation, there's a main banner for the 'MDK Microcontroller Development Kit'. It features a 'Download MDK v5.25' button and a note about an update. To the right of the banner are two columns of 'Quick Links' and 'Software Packs'. Under 'Quick Links', there are links to Getting Started, Online Manuals, Compare Editions, Middleware, and Learn. Under 'Software Packs', there are links to 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 a grid of icons representing various tools and components. The icons include MDK Tools (MDK-Core, μVision IDE & debugger with pack management), Arm C/C++ Compiler (With safety qualification), Software Packs (Device, Startup, CMSIS, MDK-Middleware, Network, Graphics, USB, Mbed TLS, File System, IoT connectors), and CMSIS Drivers (CMSIS Core, CMSIS-DSP, CMSIS-RTOS). At the bottom of the page, there are several small paragraphs of text providing detailed information about the software components.

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

- B. 현재 참에서 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

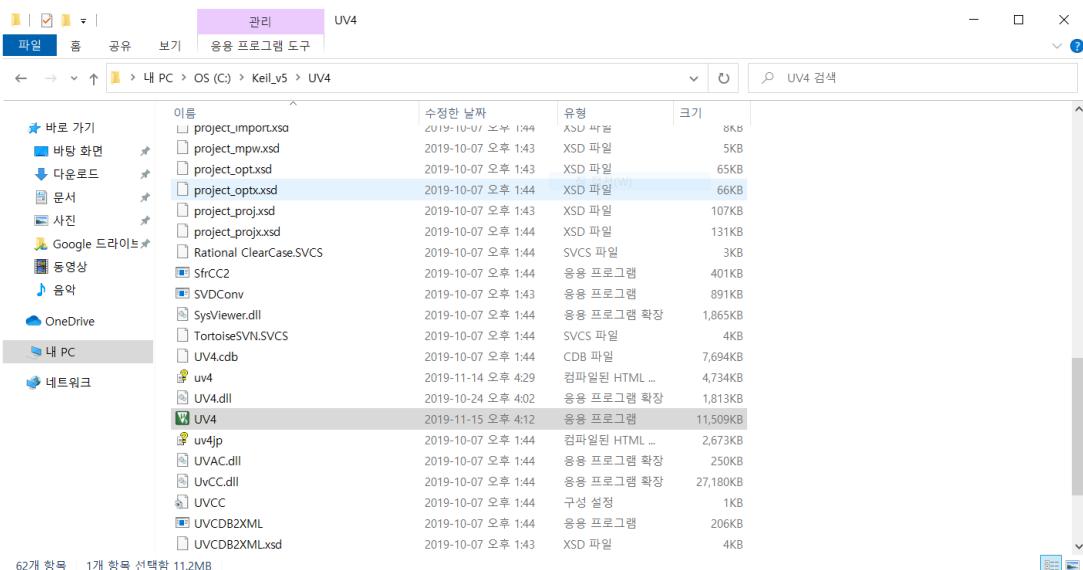
MDK Editions

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

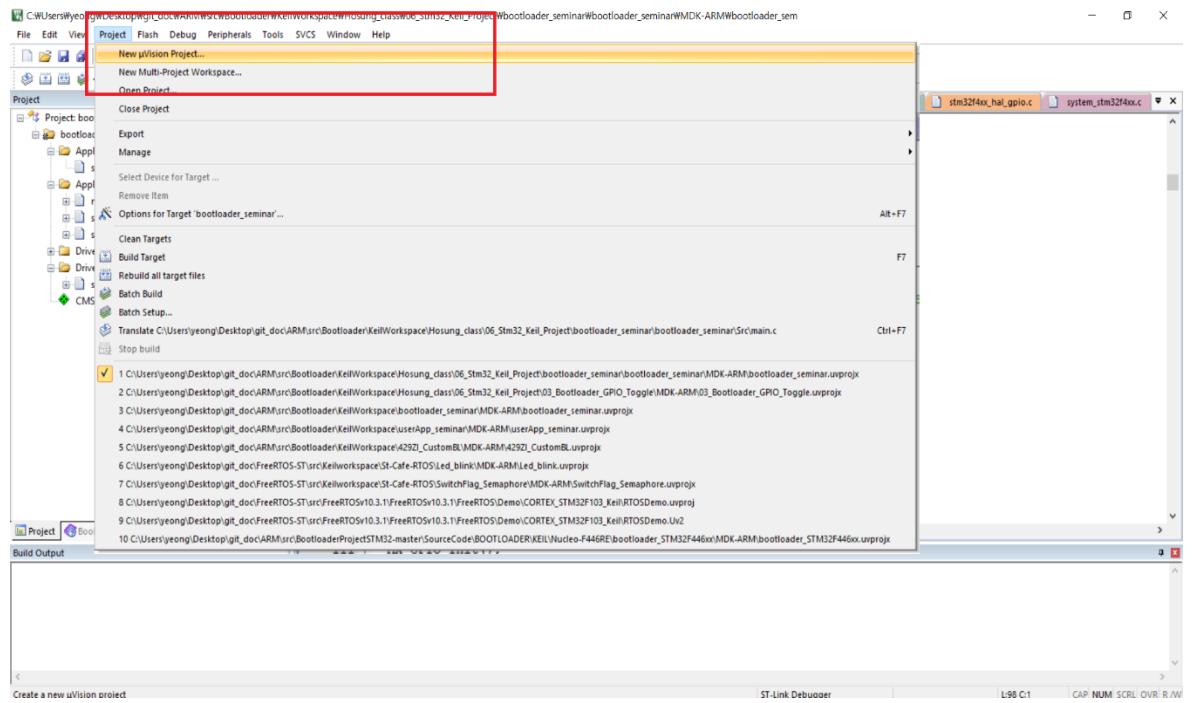
MDK-Lite	MDK-Essential	MDK-Plus	MDK-Professional
Product evaluation, small projects, and education. Code size restricted to 32 Kbyte.	For Arm Cortex-M based microcontroller projects.	For Cortex-M, Arm7, Arm9. Includes middleware (IPv4 Networking, USB Device, File System, Graphics).	For Cortex-M, Arm7, Arm9. Includes middleware (IPv4/IPv6 Networking, USB Host & Device, File System, Graphics, mbed components).
Learn more >	Learn more >	Learn more >	Learn more >
Download & Install	Request a Quote	Request a Quote	Request a Quote
	Buy online	Buy online	Buy online

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

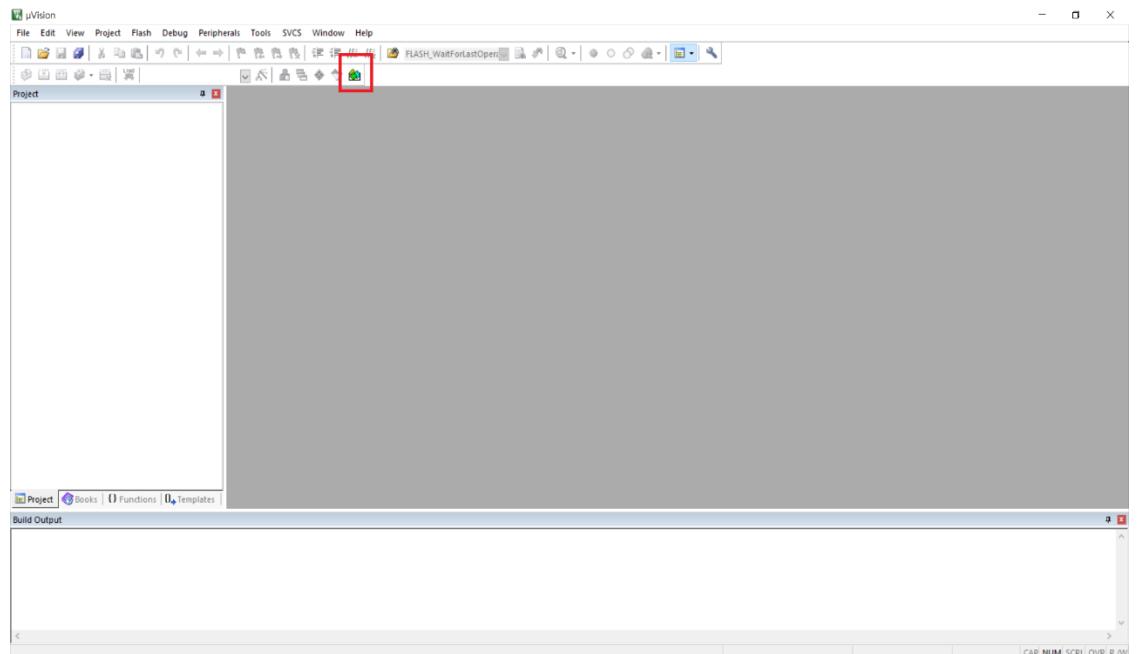
- C. 다운로드 받은 설치파일을 기본 설정으로 두고 다운로드를 시작한다. 설치가 모두 완료되면 기본 경로로 설치했으니, 다음 경로(C:/Keil_v5/UV4)로 이동하면 UV4 IDE를 찾을 수 있다. 따라서 이를 실행하도록 한다



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

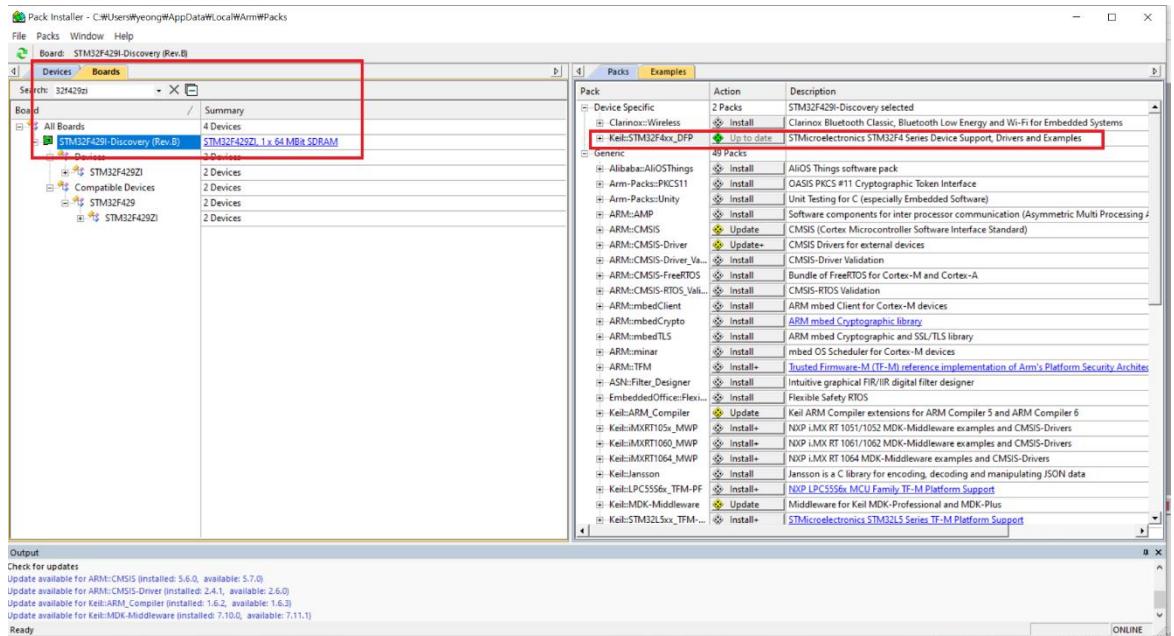


E. 하지만 프로젝트를 생성하기 위해서 Target을 잡아야 하는데, Target 파일이 모두 설치되지 않고, 원하는 Target 파일을 사용자가 직접 선정해야만 한다. 따라서 Pack Installer를 실행하도록 한다

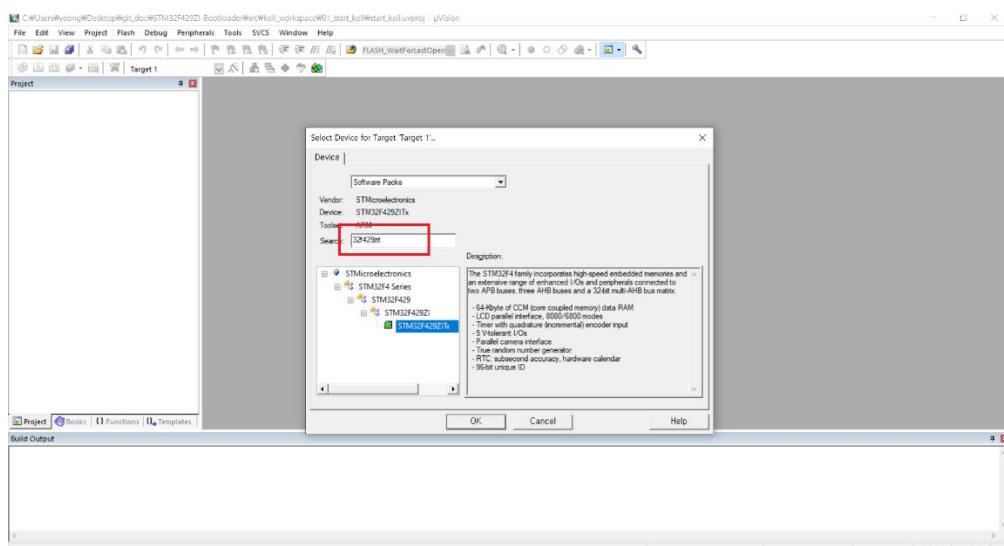


- F. Board 탭을 클릭해서 '32f429zi'를 작성하면 설치 목록이 나오게 된다. MCU 개발에 있어 기본적으로 필요한 드라이버와 startup과 같은 파일은 DFP에 담겨 있다. 그리고 보드 자체로 다운로드를 받으면 간단한 예제파일 몇개도 다운로드를 해주게 된다. 처음에는 Online 경로를 통해 패키지 목록을 받기 때문에 시간이 걸릴 수도 있다. 아래 Online에 퍼센트가 증가하는 것을 볼 수 있을 것이다

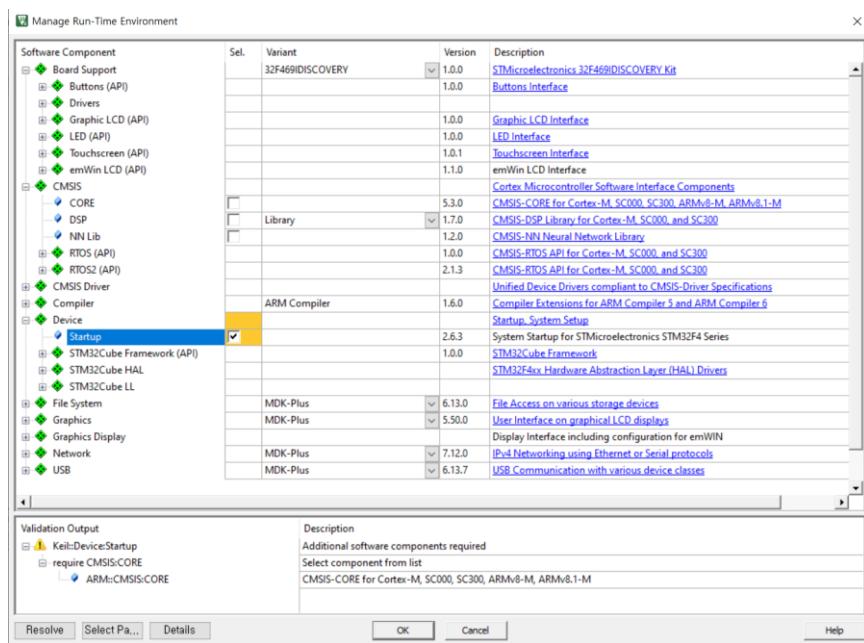
설치를 마치면 해당 패키지가 up to date로 될 것이다.



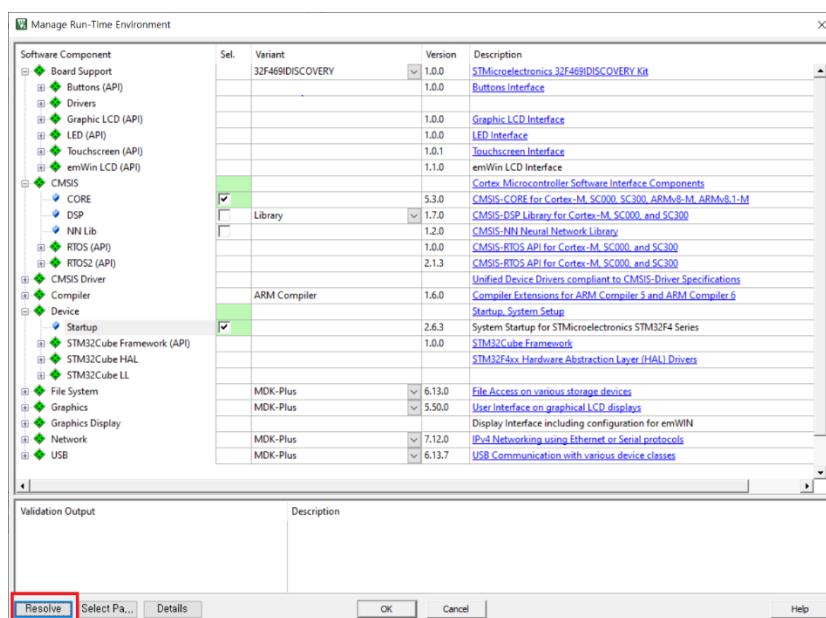
- G. 설치가 완료되었다면 pack installer 창을 끄도록 한다. 그리고 아까 하고자 했던 프로젝트 생성 버튼을 클릭하도록 한다. 그리고 search 창에 '32f429zit'을 입력하도록 한다



- 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

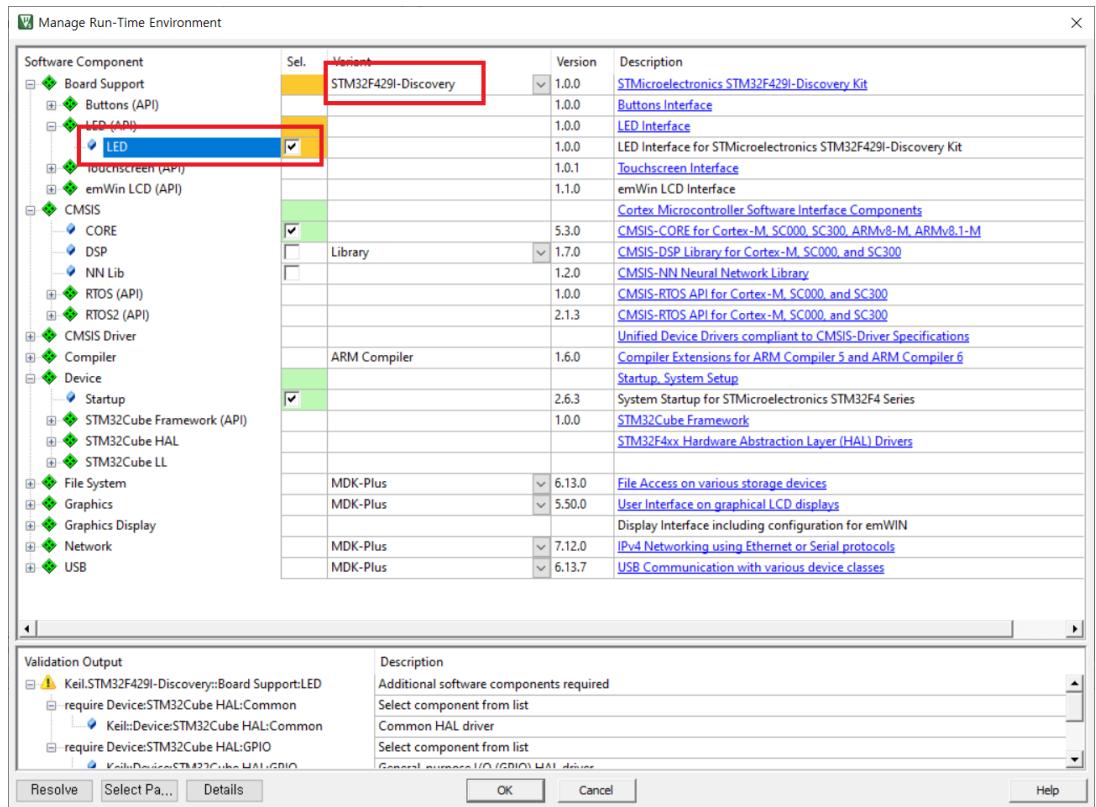


- I. 이 창에서 노란색이 뜬 것은 dependency 관련한 주의사항이다. 즉 해당 설정을 100% 완료하기 위해선 dependency를 해결해야만 한다. 이는 왼쪽 하단에 'resolve'를 눌러 쉽게 해결할 수 있다

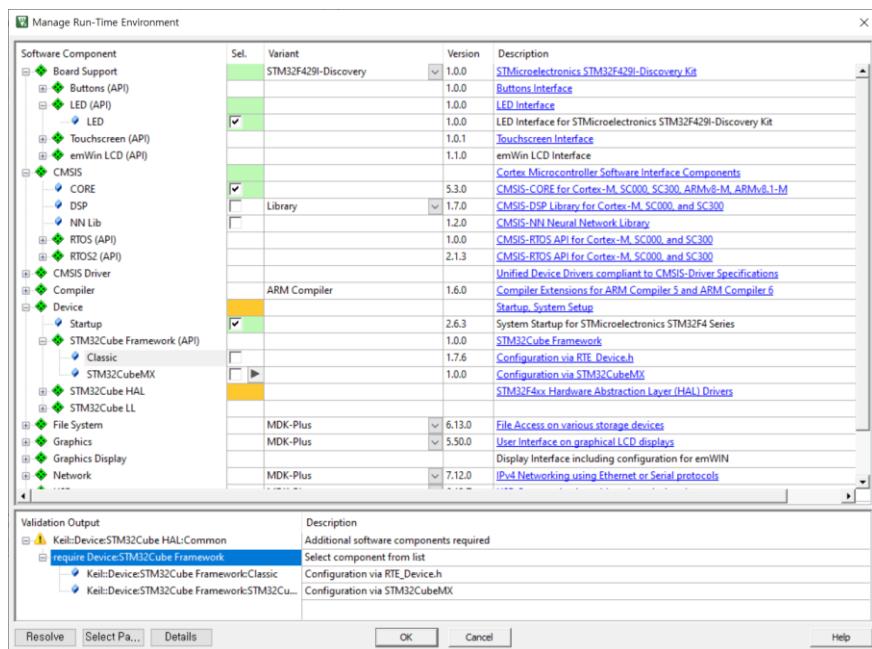


누르는 순간 CORE쪽이 체크가 되면서 초록색으로 바뀌는 것을 볼 수 있다

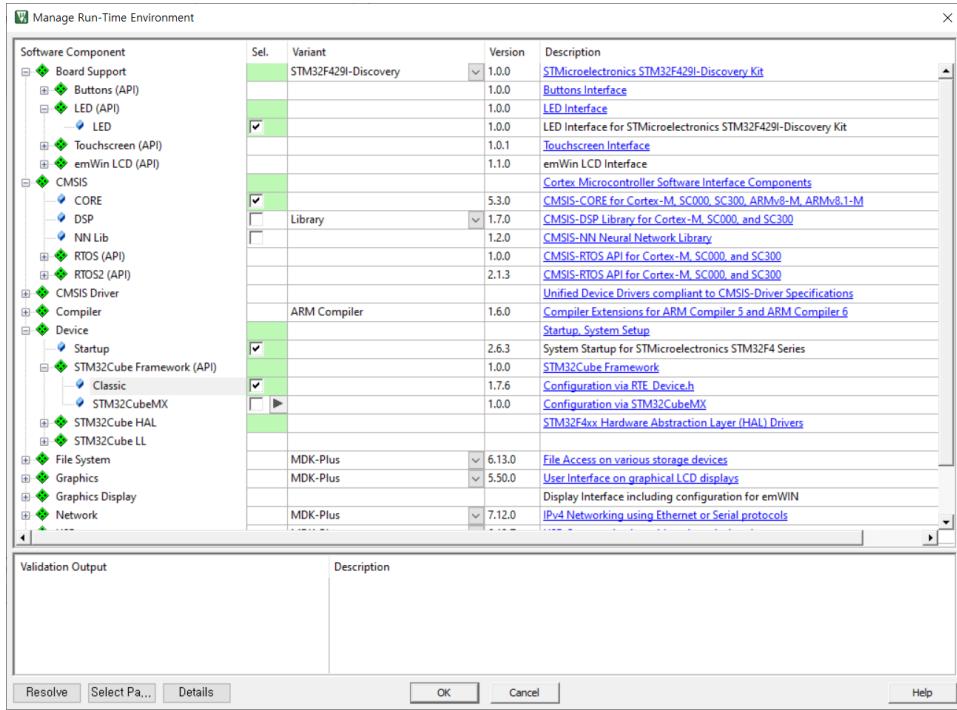
- J. 그리고 BSP로 주어지는 Led blink를 활용할 것이다. 맨 위에 Board Support 탭에서 반드시 보드를 올바르게 맞춰준 후 LED 부분을 클릭하도록 한다



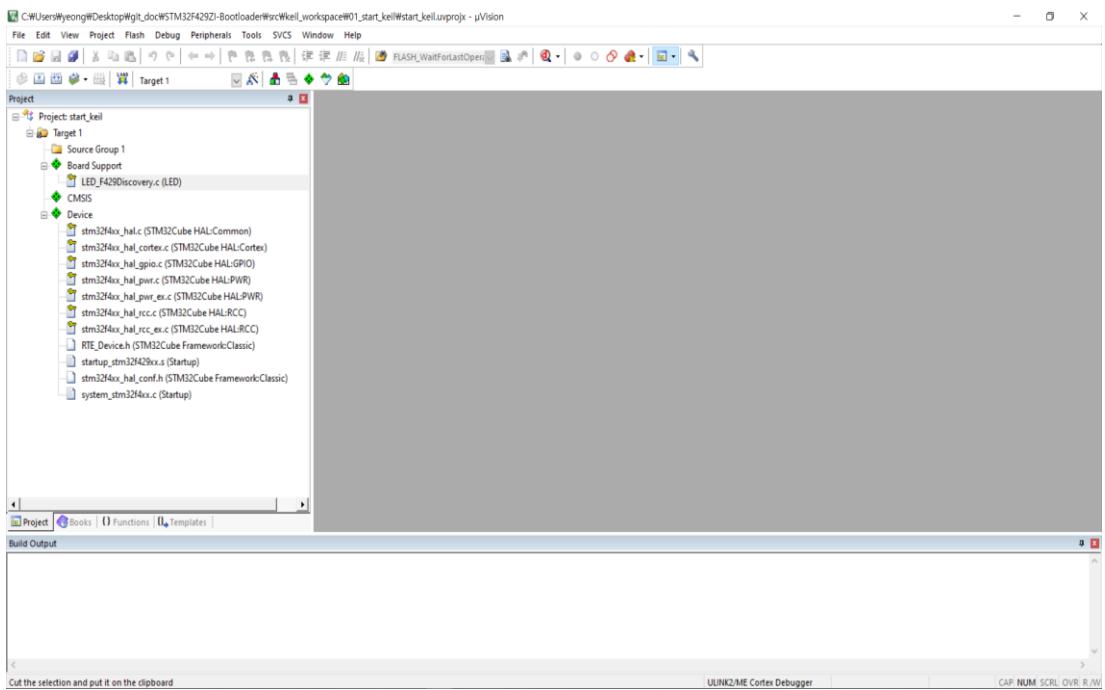
그러면 다시 dependency 주의가 나오게 된다. Resolve를 통해 해결해보도록 한다.



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

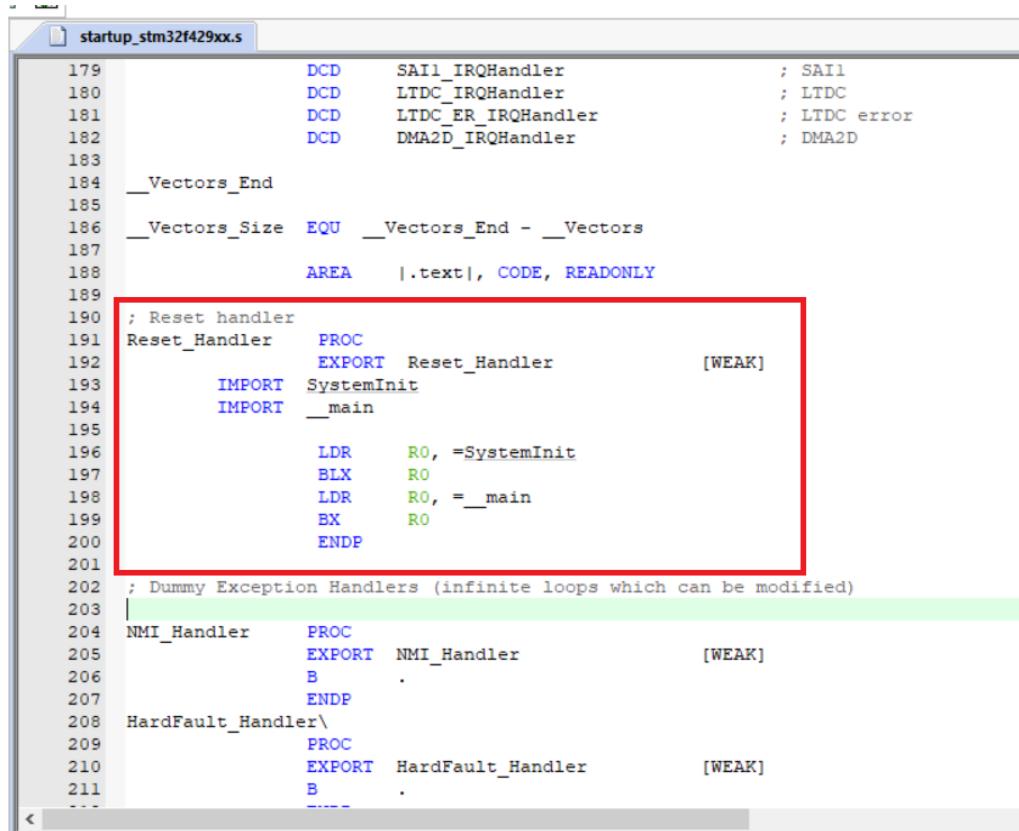
a. stack, heap 사이즈를 설정한다

```
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

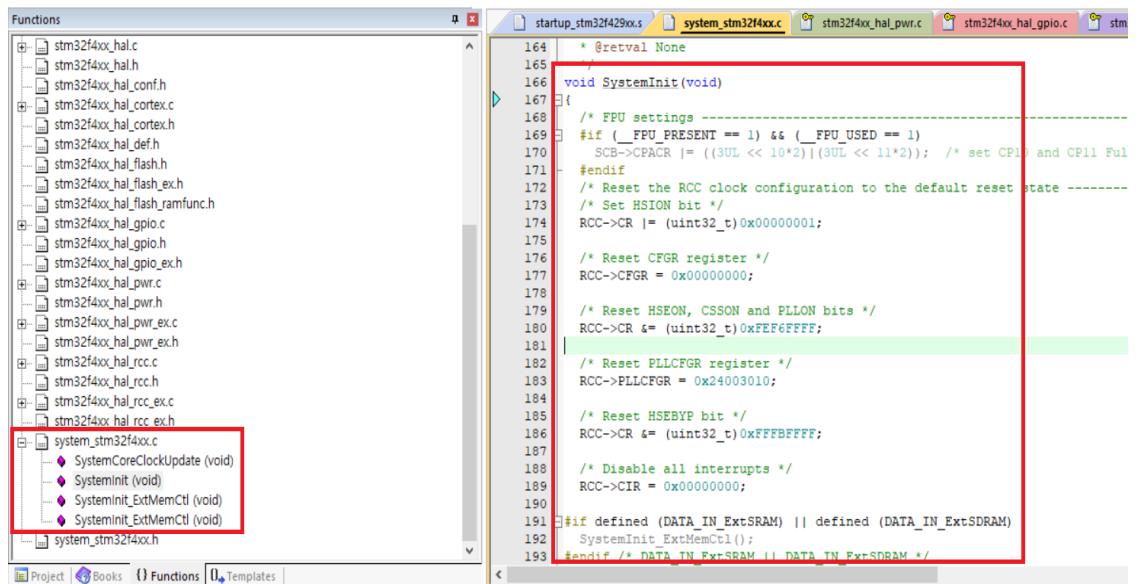


```

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**



Functions

- stm32f4xx_hal.c
- stm32f4xx_hal.h
- stm32f4xx_hal_conf.h
- stm32f4xx_hal_cortex.c
- stm32f4xx_hal_cortex.h
- stm32f4xx_hal_def.h
- stm32f4xx_hal_flash.h
- stm32f4xx_hal_flash_ex.h
- stm32f4xx_hal_flash_ramfunc.h
- stm32f4xx_hal_gpio.c
- stm32f4xx_hal_gpio.h
- stm32f4xx_hal_gpio_ex.h
- stm32f4xx_hal_pwr.c
- stm32f4xx_hal_pwr.h
- stm32f4xx_hal_pwr_ex.c
- stm32f4xx_hal_pwr_ex.h
- stm32f4xx_hal_rcc.c
- stm32f4xx_hal_rcc.h
- stm32f4xx_hal_rcc_ex.c
- stm32f4xx_hal_rcc_ex.h
- system_stm32f4xx.c**
 - SystemCoreClockUpdate (void)
 - SystemInit (void)
 - SystemInit_ExtMemCtl (void)
 - SystemInit_ExtMemCtl (void)
- system_stm32f4xx.h

startup_stm32f429xx.s system_stm32f4xx.c stm32f4xx_hal_pwr.c stm32f4xx_hal_gpio.c

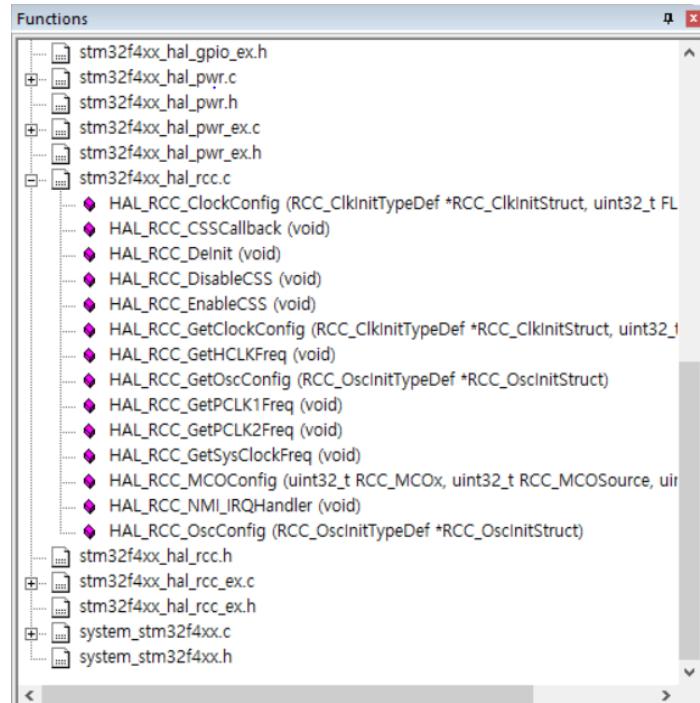
```

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

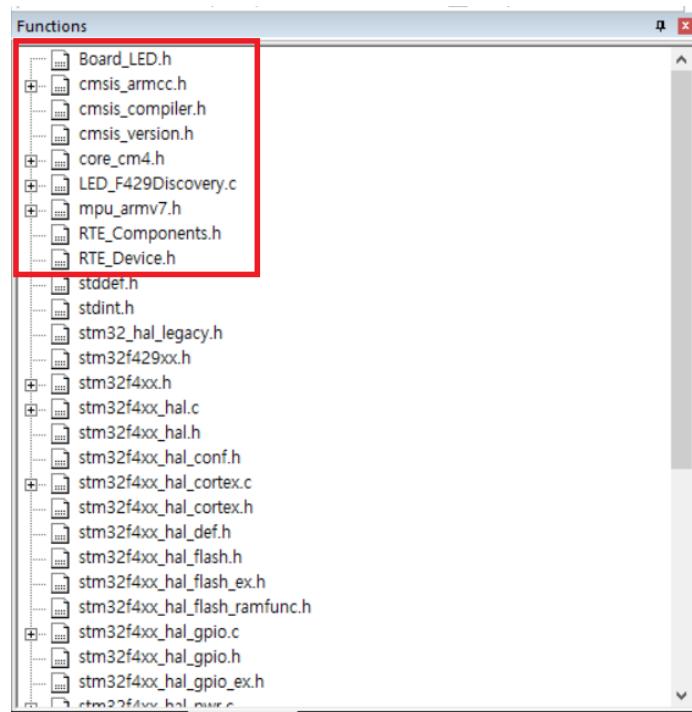
```

B. Peripheral Driver

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



C. ARM CMSIS, BSP 관련 파일들

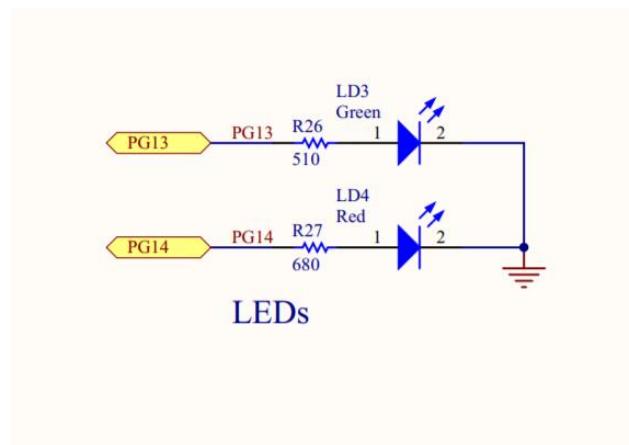


1.7 Led blink 파일 작성

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

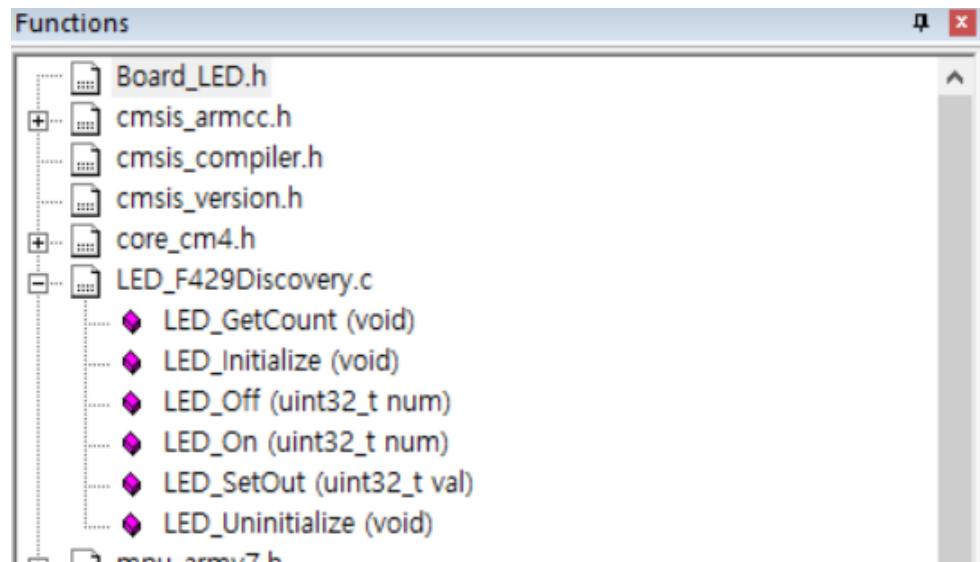
A. 회로도 확인

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



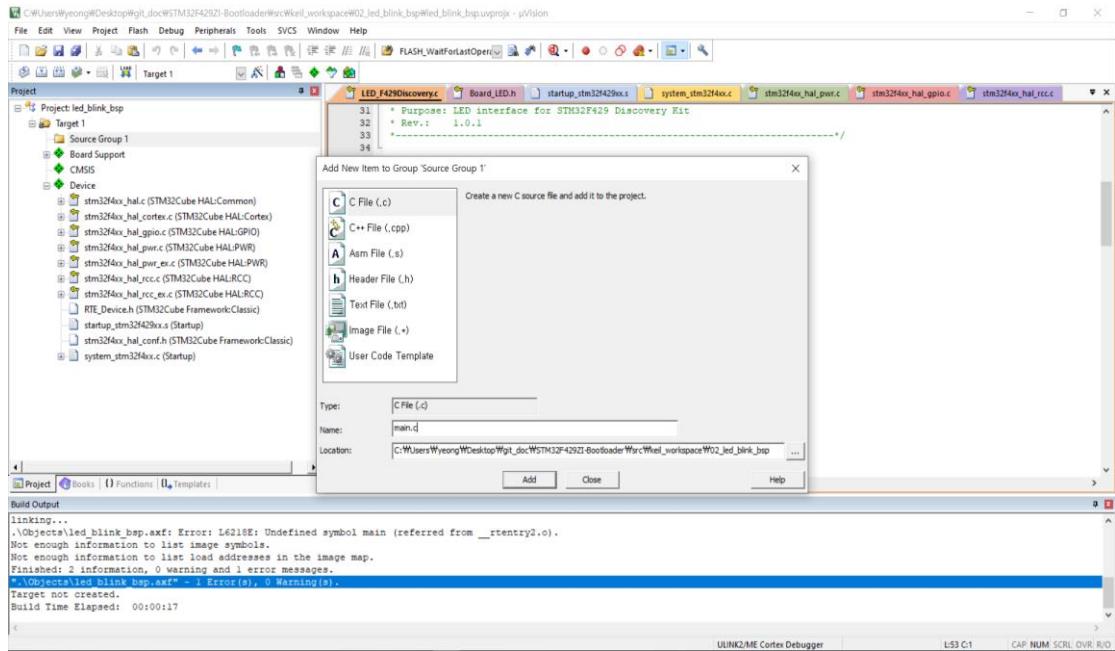
B. BSP 소스파일 확인

- 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



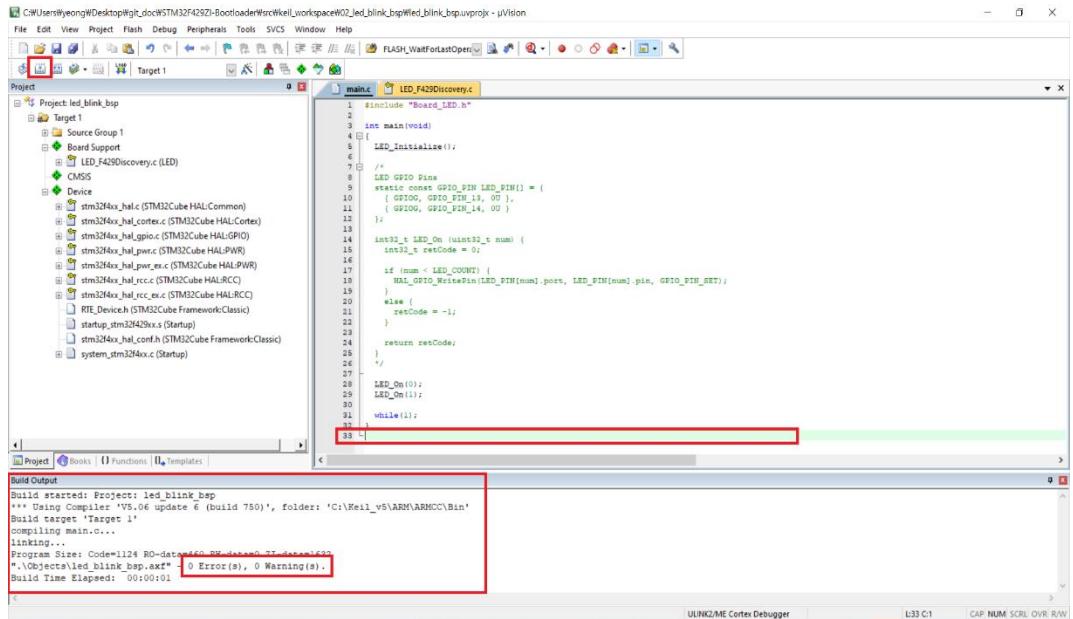
그리고 다음과 같이 소스파일을 작성할 수 있다. 소스파일에 대한 내용은 스스로 분석하는 것을 요한다.

```

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       */
10     static const GPIO_PIN LED_PIN[] = {
11         { GPIOG, GPIO_PIN_13, OU },
12         { GPIOG, GPIO_PIN_14, OU }
13     };
14
15     int32_t LED_On (uint32_t num) {
16         int32_t retCode = 0;
17
18         if (num < LED_COUNT) {
19             HAL_GPIO_WritePin(LED_PIN[num].port, LED_PIN[num].pin, GPIO_PIN_SET);
20         }
21         else {
22             retCode = -1;
23         }
24
25         return retCode;
26     }
27
28     LED_On(0);
29     LED_On(1);
30
31     while(1);
32 }

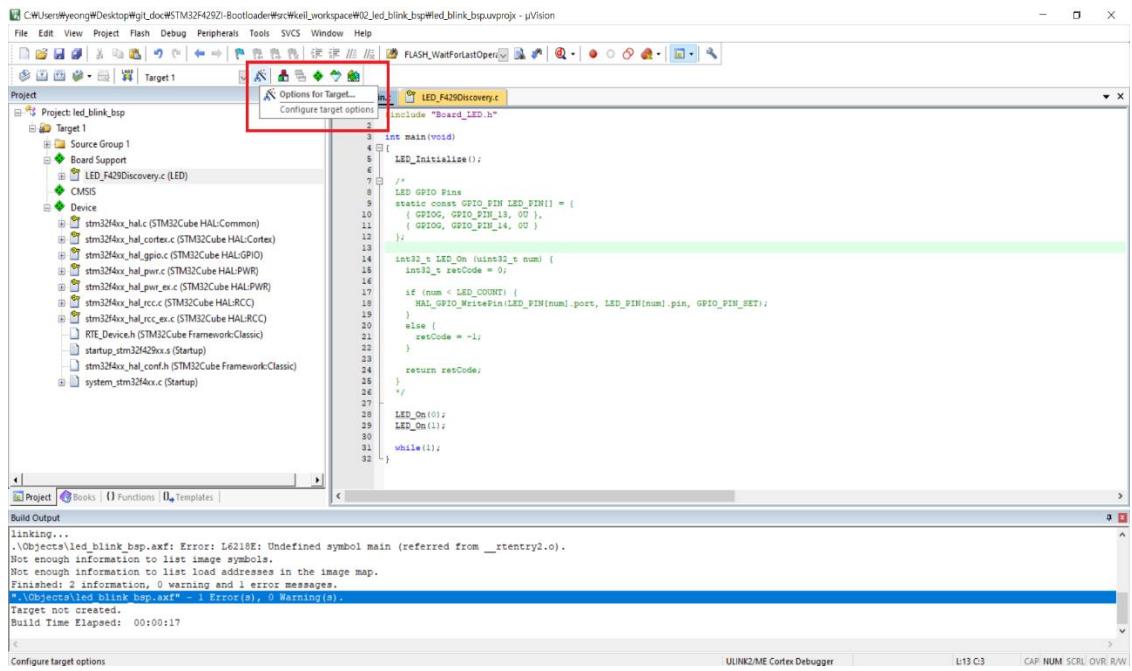
```

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

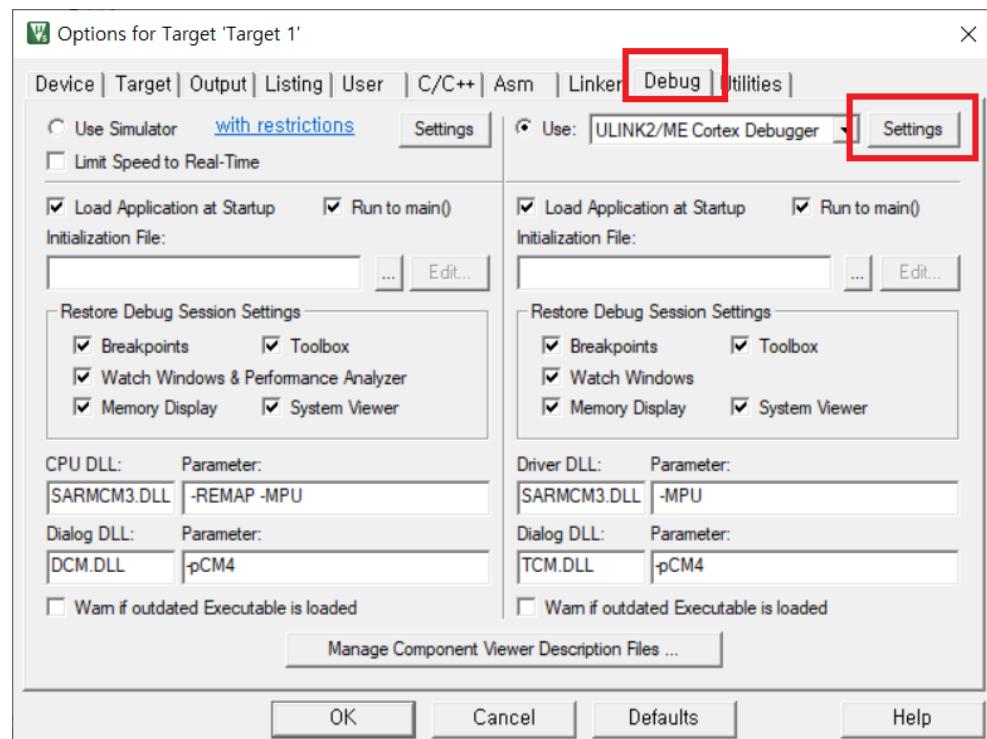


D. Target Flash Option

- 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’를 누르도록 한다



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



2. System Memory 부트로더

3. Custom 부트로더 작성

4. PC Host 통신 프로그램 작성 [Python]

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

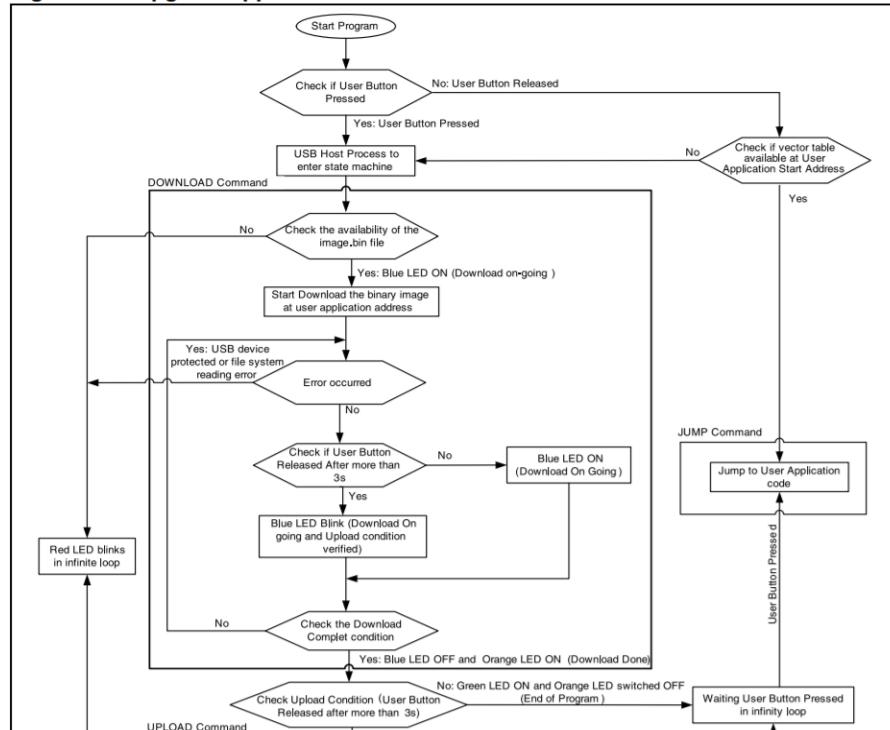
User button state	Time	Process executed
Pressed	> 3 seconds	UPLOAD command will be executed immediately after completed execution of the DOWNLOAD command.
	< 3 seconds	Only the DOWNLOAD command is executed.

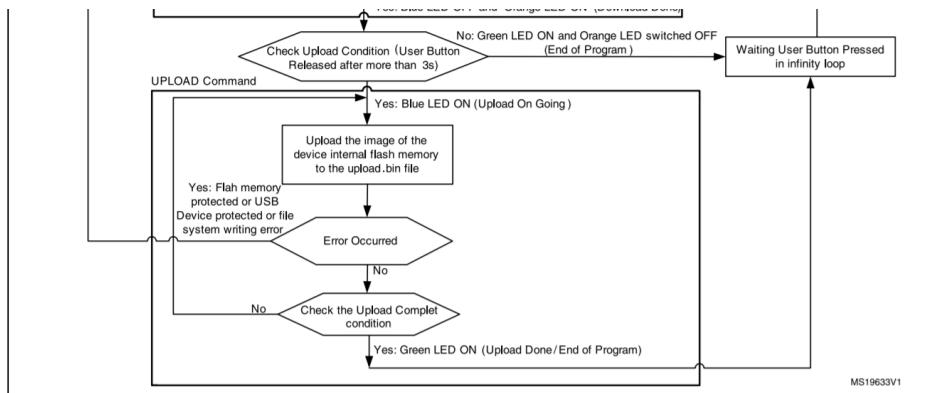
Note: The UPLOAD command condition verification is signaled by the blinking state of the blue LED.

- 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

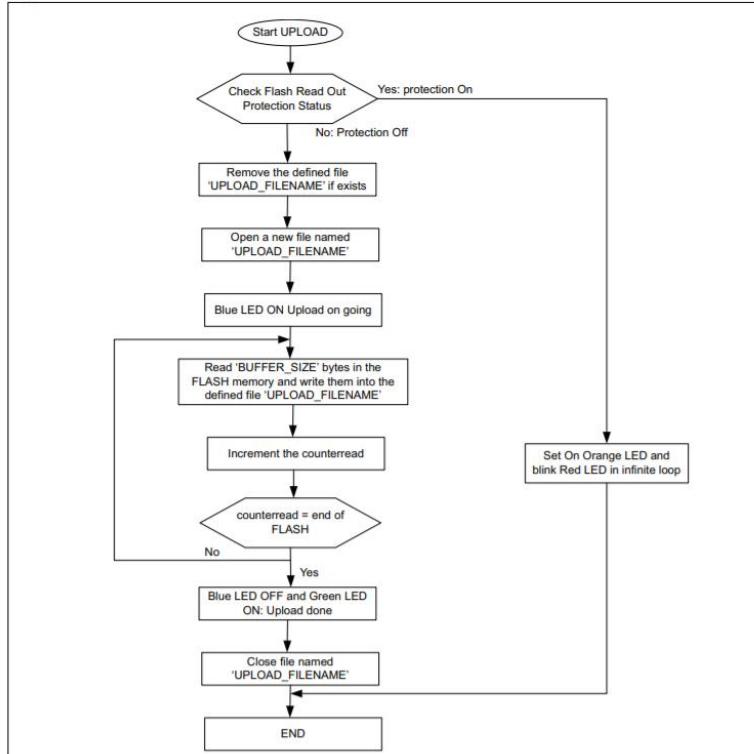




- 프로그램이 시작된다
- User 버튼이 눌렸는지 확인한다. 안눌렸으면 user program을 실행한다
- USB 초기화
- image.bin 파일이 있는지 확인한다. 있다면 blue(green) led가 켜질 것이다
- 그리고 3초 동안 delay를 가지면서 버튼이 그 이후에도 눌렸는지 확인한다
- 먼저 3초 이후 버튼이 눌린 여부는 상관하지 않고, 초기에 버튼이 눌렸다면 펌웨어 업그레이드를 진행한다
- 만일 3초 이후에도 버튼이 눌렸다면 업로드를 실행하도록 한다
- 그리고 User 어플리케이션으로 jump해서 실행하도록 한다

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

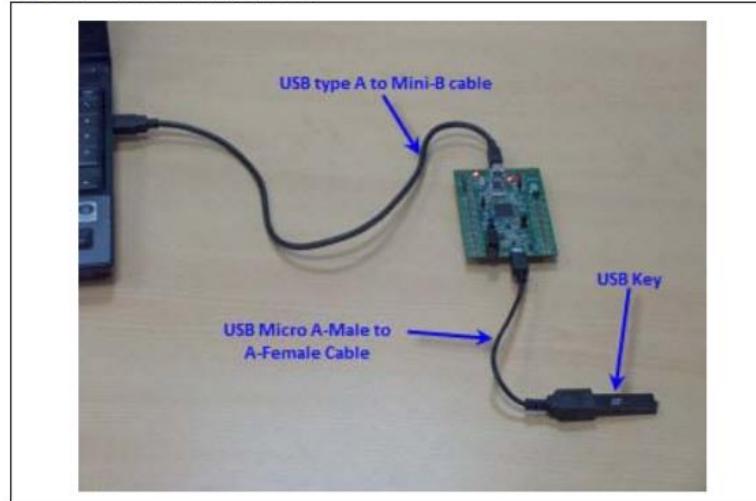
E. 시스템 요구사항

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.

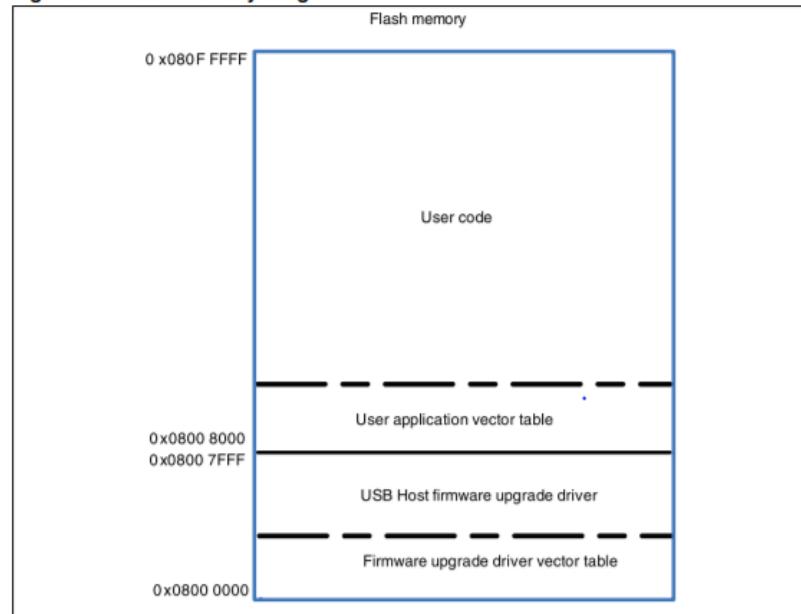
- a. 먼저 보드와 프로그램 다운을 위한 케이블이 있어야 한다
- b. 그리고 USB micro A-Male to A-Female Cable을 따로 준비해야 한다
- c. 펌웨어 업그레이드를 하기 위해서는 image.bin 파일과 담은 USB key를 가지고 있다
- d. image.bin 파일은 반드시 아래 요구사항을 따라야 한다

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



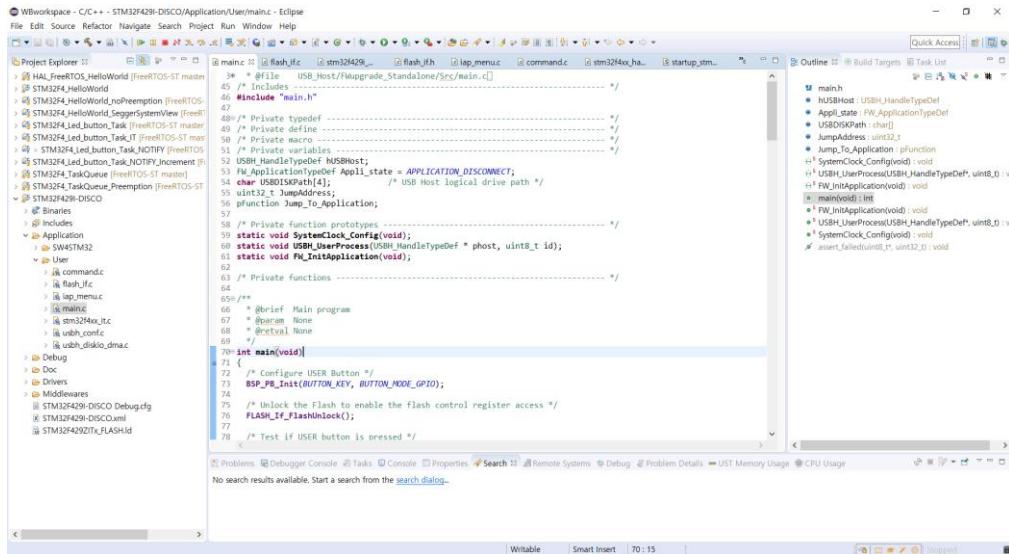
시작 address가 APPLICATION_ADDRESS와 같아야 한다. Program할 때 Linker에서 설정할 수 있다. 그리고 VECT_TAB_OFFSET을 (APPLICATION_ADDRESS 0x80000000)으로 수정해야 한다.

F. 실습

위 문서 설명과 예제는 STM32F4-Discovery 기반이다. 따라서 이어지는 다음 문서 내용과 다르게 다를 것이다. 지금부터 다룰 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))
    {
    }
```

JAP_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 do nothing */
    }
}
```

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 */
    /* Ensure completion of memory access */

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

typedef struct
{
    __IM uint32_t CPID; /*!< Offset: 0x000 (R/ ) CPUID Base Register */
    __IOM uint32_t ICSR; /*!< Offset: 0x004 (R/W) Interrupt Control and State Register */
    __IOM uint32_t VTOR; /*!< 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;

```