[DIY_Drone{}]
clock configuration.pdf

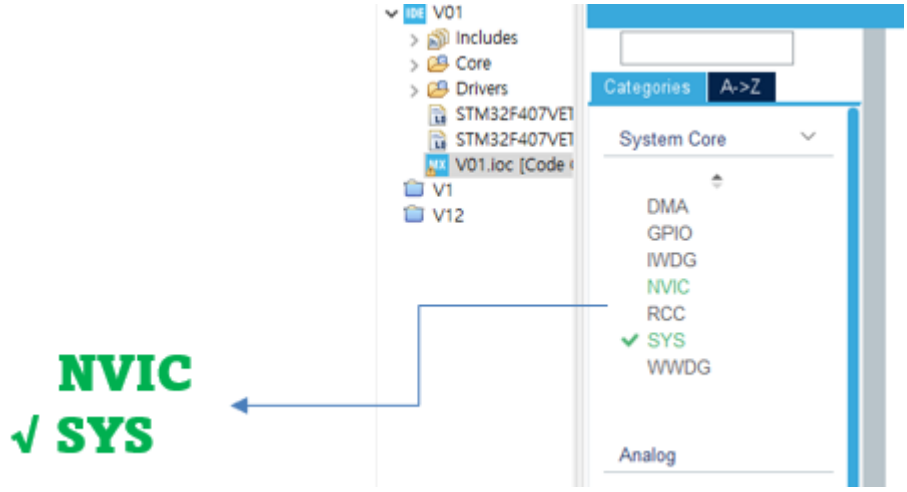
2022-06-20 지규선

[stm32CubeIDE 첫 시작]

stm32CubeIDE를 실행하고, 타겟보드를 stm32F407VET로 하여 프로젝트를 생성하면, 관련 라이브러리가 다운로드되는데요, 그 시간이 생각보다 길어요.

[처음에는 SysTick만 활성화 되다]

[Pinout & configuration] 메뉴의 [카테고리]에서 [System Core]를 열면, 엔빅은 녹색으로, 시스는 사용중으로 체크되어 있습니다.



이는 시스틱을 시스템클럭으로 사용하겠다는 뜻인데요, 시스틱은 인터럽트 방식을 이용하므로 엔빅이 동행합니다. 여기서, 엔빅은 '인터럽트 매니저'를 의미해요.
이 때, 시스와 엔빅 외에는 모두 검은색인데요, 앞으로 사용자가 생명을 부여해야 할 대상들이란 뜻을 포함하고 있습니다.

이렇듯, 프로젝트가 새롭게 생성되면 기본적으로, CPU 내장형 클럭인 시스틱만 활성화되고, 나머지는 모두 비활성 상태로 준비됩니다.

[SysTick의 한계]

[Clock Configuration]을 보겠습니다. 정말, 시스틱을 사용하는 내장형 클럭 두 개만 활성화되어 있네요. Low Speed Internal은 저속 내장형 클럭, High Speed Internal은 고속 내장형 클럭입니다.

32KHz 저속 내장형 클럭은 주로 RTC용인데요, RTC는 시각이나 알람을 구현하기 위한 사전 정의형 타이머를 말합니다. 참고로, RTC 클럭 선택기는 [Timers]에서 RTC 타이머를 설정해야 활성화됩

니다.

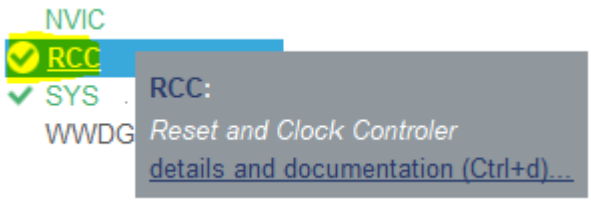
16MHz 고속 내장형 클럭은 시스템클럭으로 바로 사용될 수 있는데요, 그 라인을 따라가 볼까요?

두 갈래 길이 있네요.

첫째, 고속 내장형 클럭이 시스템클럭 선택기로 바로 가면 시스클럭, 즉 시스템클럭은 그대로 16MHz가 됩니다.
둘째, 고속 내장형 클럭이 PLL 소스 선택기로 우회하면 시스템클럭은 최대 168MHz가 됩니다. 여기서, PLL은 일단, '주파수 증폭기' 정도로 이해하고 넘어가세요.

그런데, 16MHz 클럭은 너무 작아요. 고성능인 stm32F407VET 보드를 일부러 저성능으로 다운그레이드시킬 필요는 없겠지요. 그래서 일딴은 PLL 회로를 거친 PLL클럭(PLL 클럭), 즉 168MHz를 시스템클럭으로 삼습니다.
그런데 말이죠, CPU 내장형 클럭, 즉 시스틱과 관련된 클럭은 권장 사항이 아니라네요. 잠시 그 근거를 확인해보겠습니다.

[Pinout&Configuration] 메뉴의 [카테고리] 중, [System Core]를 열어 RCC를 선택합니다. 마우스를 그 위에 올리면 매뉴얼 선택기가 뜹니다.



Ctrl + d를 누른 다음, 참고매뉴얼을 선택하면 관련 PDF 파일이 열리는데요, 이 문서에서 HSI Clock을 검색합니다. 검색 단축키는 CTRL + F 입니다.

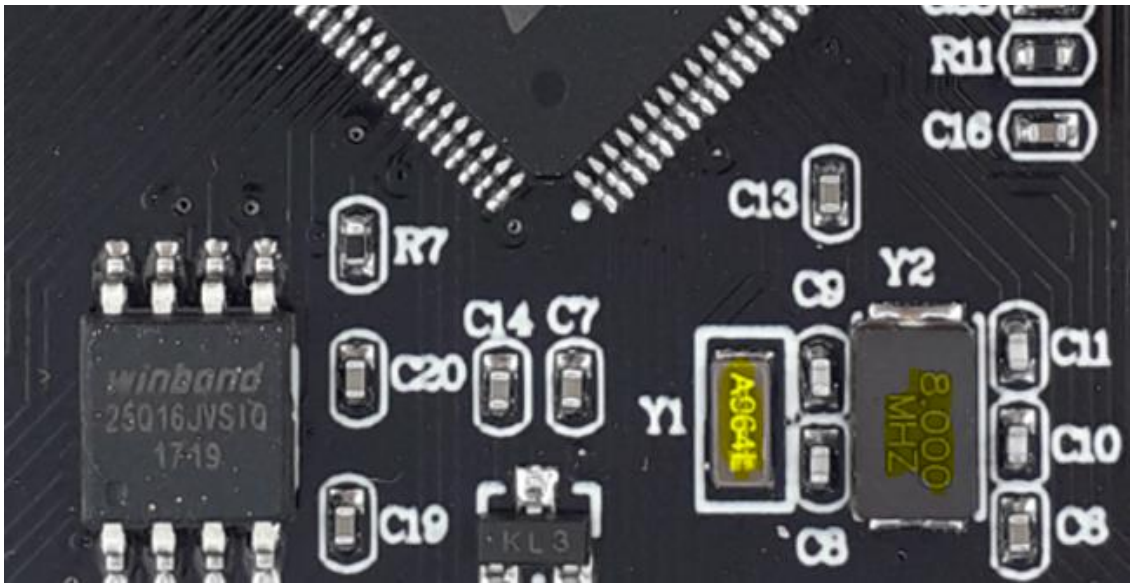
문서 내, 7.2.2 HSI clock을 보면,

"CPU 내장형이므로 기동 시간이 빠르고 경제적이지만, 외부 오실레이터에 비해 정확도가 떨어진다! "

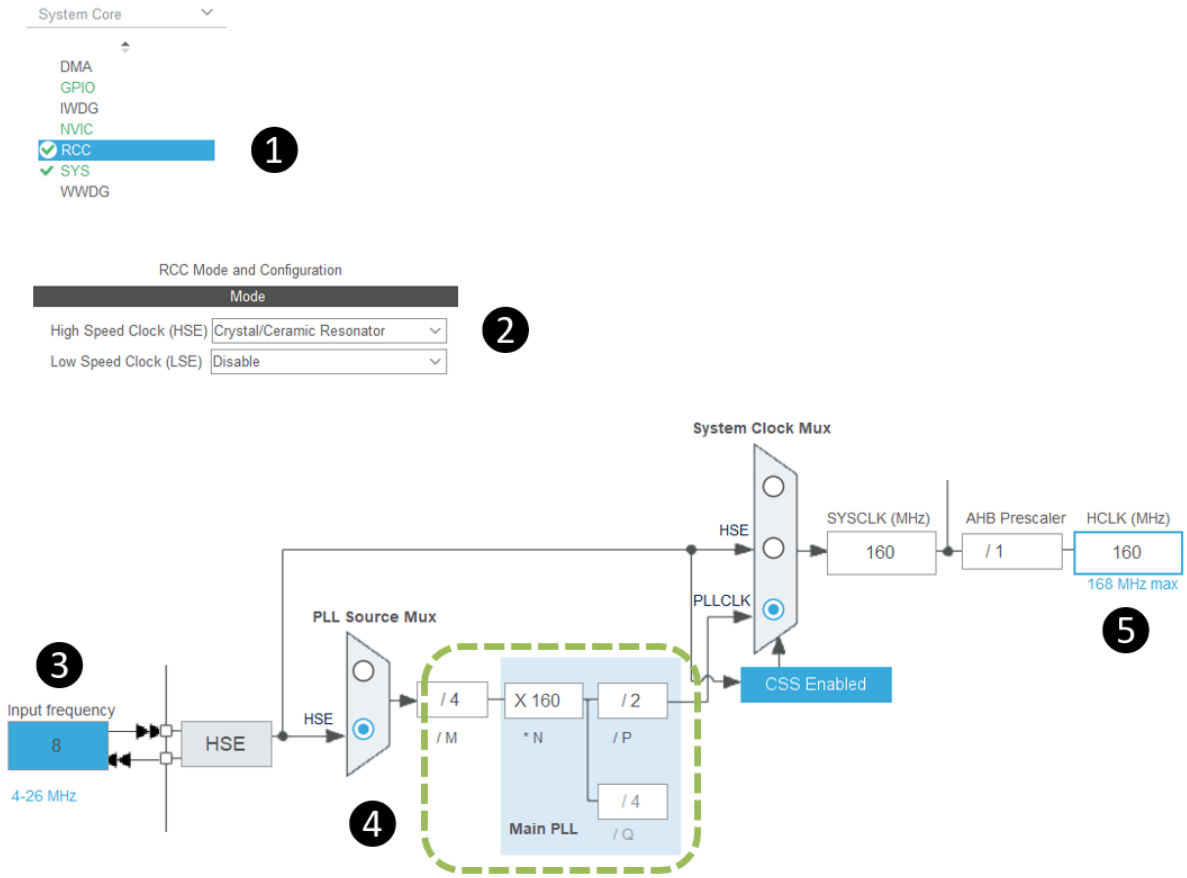
아마도 CPU 발열 때문이라고 생각되는데요, 이에 반해, 외장형은 말그대로 CPU 외부에 존재하므로 열과는 관련이 없어요. 성능 좋은 FC를 만들려면 결국, 외장형 클럭을 사용하라는 말이네요.

[외장형 오실레이터를 이용하여 시스템클럭 만들기]

stm32F407 보드에 외장형 클럭 발생기가 있어야 할텐데요, 한번 볼까요?



32.768KHz 저속 외장형 오실레이터와 8MHz 고속 외장형 오실레이터가 나란히 붙어 있군요. 이 중에서 우리는 주로 8MHz 고속 외장형 공진기를 사용할텐데요, 지금부터는 시스템클럭이 어떤 과정을 거쳐 만들어지는지 살펴보겠습니다



첫째, [Pinout&Configuration]의 카테고리 중 [System Core]에서 [RCC]를 선택합니다.
둘째, [Mode]창에서, 고속 외장형 클럭을 Crystal/Ceramic 공진기로 설정합니다.
셋째, [Clock Configuration]으로 이동하여, 새롭게 활성화된 고속 외장형 공진기 주파수 입력란에 8을 입력합니다. 참고로, H7 시리즈는 25MHz 공진기를 사용해요.
넷째, 어느 경로를 거칠지 정합니다.

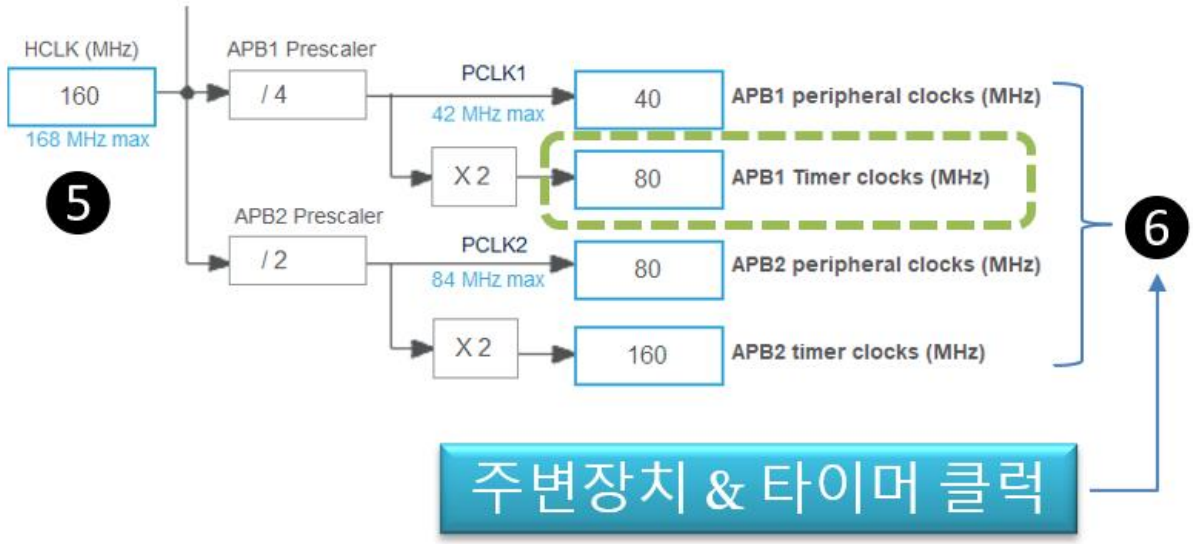
시스템클럭 선택기로 바로 가거나, PLL 소스 선택기로 우회해서 가는 길, 역시 두 갈래 길이 존재하네요. 시스템클럭 선택기로 바로 가면 시스템클럭은 최대 8MHz, PLL 소스 선택기로 우회하면 최대 168MHz가 됩니다. 우리가 원하는 것은 고성능 FC이므로, 당연히 PLL 회로를 거쳐야 하겠죠?

[PLL 회로]

PLL 회로를 보겠습니다.
외장형 공진기의 주파수 8MHz가 PLL 소스 선택기로 들어가면, 증폭되어 PLLCLK 168MHz로 변환되고, 바로 시스템클럭 선택기로 보내지는데요, 이 때, 시스템클럭 선택기는 CSS(Clock Security System)를 활성화시켜 PLLCLK을 검증합니다. CSS는 Clock Security System의 줄임말로서, 외장형 공진기에서 들어온 주파수를 검증합니다. 참고로, 내장형 클럭에는 CSS가 필요없어요. 내부인은 무사통과, 외부인은 통제, 뭐 그런 시스템인 셈이죠. 이제, 시스템 클럭 선택기의 보안시스템을 통과한 PLLCLK은 그대로 시스템클럭이 됩니다.

[주변장치 및 타이머용 클럭 만들기]

이제, 168MHz 시스템 클럭을 타이머 및 주변장치용 클럭으로 최종 전환시켜야 합니다. 우리가 필요한 클럭은 시스템클럭이 아니라, 타이머와 주변장치용 클럭이거든요.



그전에, 시스템클럭을 160MHz로 바꾸고 시작해야 합니다. 제 3강에서 실제 사례를 들어 설명할 테지만, 일단은 '프리스케일링의 편의'를 위해서라고 생각하세요. 시스템클럭의 최대 속도, 즉 최대 주파수가 168MHz이므로, 그보다 낮은 160MHz로 설정해도 괜찮습니다. 더욱 낮추어, 80MHz로 설정해도 무관하고요. 사실, 상식적으로 생각해도, 최대치보다는 조금 낮추어 잡는 것이 더 안정적일 수 있어요. 그래도, 168MHz를 다 사용하는 것이 일반적이긴 합니다.

시스템 클럭을 타이머 및 주변장치용 클럭으로 다운스케일하는 것은 CPU 리소스의 적정 배분이라는 측면에서 매우 중요합니다. 그런데, 그 과정에 동원되는 용어들은 마음에 쉽게 다가오지 않아요. 그래서 비유를 들곤 하는데요,

AHB는 큰 길, APB는 작은 길입니다. 작은 길은 큰길에서 분기됩니다. 곁가지가 나무 기둥에서 분기되는 것과 같아요. 큰 길이든 작은 길이든 흐름을 제어하는 이동용 바리케이트가 모두 존재하는데요, 그게 바로 PSC예요. 만일 AHB의 PSC가 1이면 큰 길이 완전히 열린 것이고, APB1의 PSC가 4이면 작은 길의 흐름이 4분의 1로 제한된다는 것을 의미해요. 참, PSC는 프리스케일러의 줄임말이예요.

HCLK는 큰 길인 AHB를 오가는 클럭으로서, 시스템클럭을 한번 프리스케일한 것입니다. 그 프리스케일러, 즉 PSC가 1이면 HCLK은 시스템클럭과 같아져요. PSC가 2이면 HCLK은 시스템클럭의 2분의 1이 되겠죠? 여기서, HCLK는 메인클럭으로 불리기도 합니다.

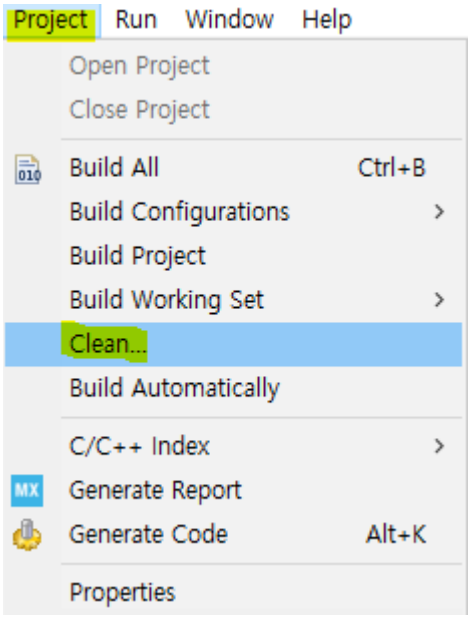
HCLK이 오가는 길인 AHB에서 분기된 오솔길은 모두 다섯 갈래입니다. 그 중에서 우리는 두 길, 즉 APB1, APB2를 주로 다룰텐데요, 바로, 타이머와 주변장치용 클럭들이 이동하는 길들이기 때문입니다. 큰 길을 오가는 시스템클럭의 주파수는 크고, 작은 오솔길을 오가는 타이머와 주변장치용 클럭의 주파수는 상대적으로 작습니다.

타이머와 주변장치용 클럭은 HCLK를 APB1과 APB2의 바리케이트인 PSC, 즉 4와 2로 나누어 얻습니다. 여기서는 4와 2을 사용했지만, 사용자의 선택에 따라 달라질 수 있는 숫자예요. 한번 자유롭게 변경해보세요. 그럼 타이머와 주변장치용 클럭의 속도, 즉 주파수가 변하는 것을 볼 수 있어요. 혹시 빨강박스가 생기면 다시 시도하면 됩니다. 마지막에는, PSC는 원래값, 즉 4와 2로 꼭 되돌려주셔야 합니다. 참고로, PSC를 다운스케일러로 생각하면 편해요.

[코드 생성]

지금까지, 시스템클럭의 주파수를 만들고, PSC를 이용해서 타이머 및 주변장치용 주파수들도 만들었습니다. 이제 우리가 할 일은, 타이머나 주변장치들을 자유롭게 사용하는 것일텐데요, 그 활용법은 해당 타이머와 주변장치가 등장할 때 다루겠습니다.

이제 저장하고,
[Project] - [Clean] 합니다. 그럼, 지금껏 설정한 내용들이 코드로 만들어져요.



그리고 main.c 파일로 이동해서 새롭게 생성된 코드를 살펴보세요.
해당 함수 위에서 CTRL + 왼마우스 클릭하면 그 함수 내부로 이동합니다.

High Speed External 선택, PLL 회로를 통해 산출된, PLL클럭의 이용, 큰 길과 작은 길들의 PSC 설정값들이 그대로 담겨 있을거예요. 참, 프리스케일러는 다운스케일러 뿐만아니라 디바이더로도 불립니다. (The end)