한화비전

# ARM Architecture

# ARM(Advanced RISC Machine)



## ARM

- 1985년 에이콘 컴퓨터에서 고안
- 1990년 ARM Limited 설립
- 양국 기반의 합작 투자 회사 – 에이콘 컴퓨터, 애플 컴퓨터, VLSI 테크놀로지

## ARM 시장 동향

- 스마트 폰의 95%
- 태블릿 시장의 85%
- 웨어러블 디바이스 90%
- 2023년 기준 305억8300만대 (누적 2500억 개)의 ARM이 탑재된 칩 출하
  https://www.seminet.co.kr/channel_micro.html?menu=content_sub&com_no=792&category=product&no=10067
- 2025년 상위 하이퍼스케일링 컴퓨팅 50%가 ARM 기반으로 예상
  https://www.e4ds.com/sub_view.asp?ch=2&t=0&idx=20660


## ARM 프로세서 IP(Intellectual Property) 형태로 제공, 반도체 제조회사 및 SoC 제조사에서 제품 생산 및 판매

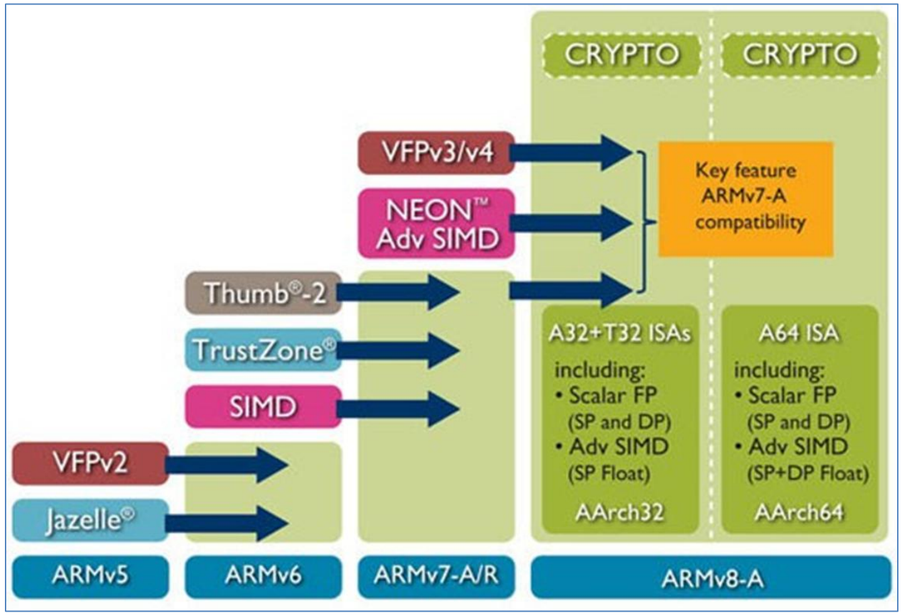
# ARM(Advanced RISC Machine)



## RISC vs CISC

| RISC (Reduced Instruction Set Computer) | CISC (Complex Instruction Set Computer) |
|---|---|
| – CPU 명령어의 개수를 줄임, 파이프라인<br>– H/W 구조를 간단하게 함<br>– 모든 연산을 하나의 클럭으로 실행<br>– 명령어 길이가 고정<br>– CISC 대비 소모 전류가 적다 | – Program을 위한 수많은 명령으로 구성<br>– H/W 구조 복잡함<br>– 복잡한 명령의 집합을 가짐<br>– H/W 스택을 지원<br>– RISC 대비 소모 전류가 많다 |

## ARM 프로세서 계보




〈출처〉 https://namu.wiki/w/Arm%20%EC%95%84%ED%82%A4%ED%85%8D%EC%B2%98

# ARM(Advanced RISC Machine)



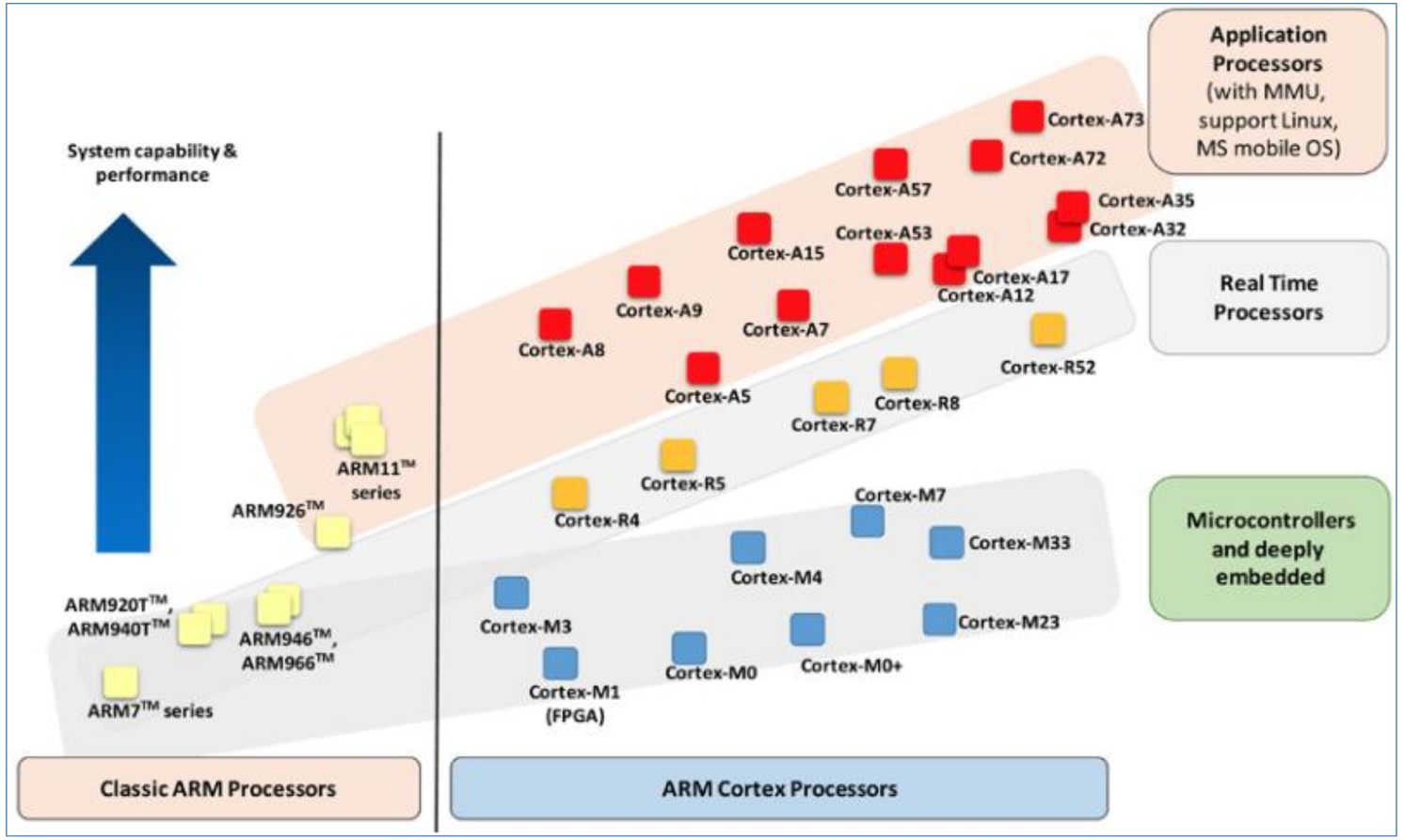
## ARM 프로세서 종류

| 프로세서 종류 | 특징 | 활용 분야 | 대표적인 3'd party |
|---|---|---|---|
| Cortex-A (application) | OS를 지원하는 고 성능 프로세서, 고 성능이나 가격이 비쌈 | 스마트 폰, 멀티미디어 디바이스, 클라우드 서버 및 데스크탑 PC 등에서 사용 | Qualcomm (Snapdragon)<br>Apple (A8, A9, A10, A11)<br>Samsung (Exynos)<br>Broadcom (BCM2835/2837/2711/2712)<br>HiSilicon (Kirin620, Huawei) |
| Cortex-R (realtime) | 높은 성능, real-time 어플리케이션에 주로 사용 | 실시간 처리에 특화된 로봇 및 항공시스템 분야에서 사용 | Texas Instrument TMS570 시리즈<br>Infineon FCR4 시리즈 |
| Cortex-M (microcontroller) | 가성비가 우수한 마이크로 콘트롤러, 낮은 전력 소모와 저렴한 가격 | IoT 기기, 전원 관리, 임베디드 오디오, 산업 및 가정 자동화, 의료 및 웰빙 어플리케이션 분야에서 사용 | STMicroelectronics (STM32Fxxx)<br>NXP (LPC1x00)<br>Atmel (SAMxx)<br>Texas Instruments<br>Freescale<br>Apple (A9)<br>Samsung (Exynos) |

# ARM(Advanced RISC Machine)



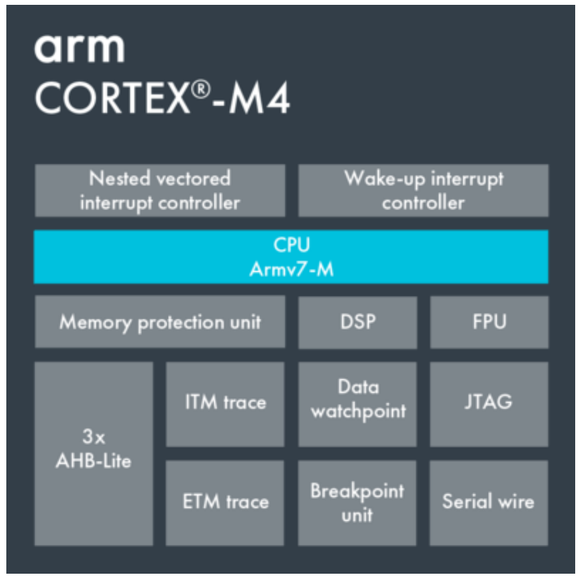
## ARM 프로세서 종류



# ARM(Advanced RISC Machine)



## □ ARM Coretex-M4




〈출처〉 https://developer.arm.com/Processors/Cortex-M4

<문서>
Arm Cortex-M4 Processor Datasheet
Cortex -M4 Devices Generic User Guide

| 특징 | Cortex-M3 | Cortex-M4 | Cortex-M33 |
|---|---|---|---|
| 아키텍처 | ARMv7-M | ARMv7E-M | ARMv8-M Mainline |
| DSP 확장 | 없음 | 있음 | 있음 |
| 부동소수점 연산장치 (FPU) | 없음 | 있음 (단정도) | 있음 (단정도) |
| 인터럽트 처리 | NVIC | NVIC | NVIC |
| 전력 소비 | 낮음 | 낮음 | 낮음 (향상된 보안 기능 포함) |
| 성능 (DMIPS/MHz) | 1.25 | 1.25 | 1.5 |
| 보안 기능 | 없음 | 없음 | TrustZone 지원 |
| 적용 분야 | 저전력 임베디드 시스템 | 신호 처리, DSP 작업, 저전력 IoT | 보안이 중요한 IoT 및 임베디드 시스템 |

# ARM(Advanced RISC Machine)



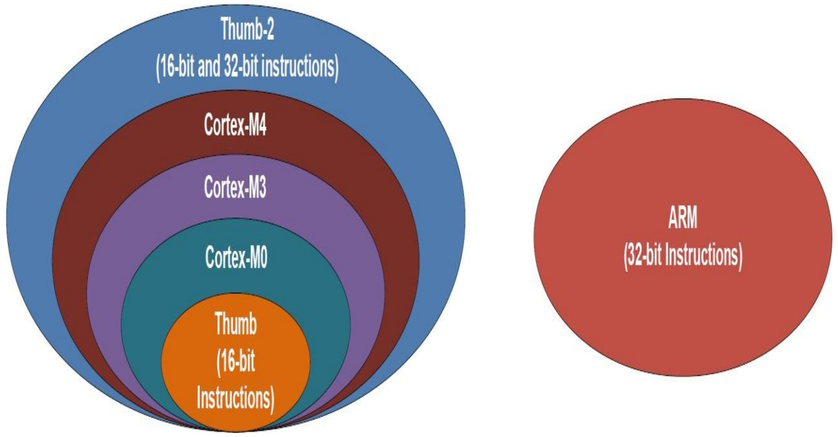
## ARM Coretex-M4 코어의 주요 기능

- Armv7E-M 아키텍처
- 버스 인터페이스 3x AMBA AHB-lite 인터페이스(Harvard 버스 아키텍처) CoreSight 디버그 구성 요소용 AMBA ATB 인터페이스
- Thumb/Thumb-2 하위 집합 명령어 지원
- 3단계 파이프라인
- DSP 확장: 단일 주기 16/32비트 MAC, 단일 주기 이중 16비트 MAC, 8/16비트 SIMD 산술, 하드웨어 분할(2-12 주기)
- 옵션 단정밀도 부동 소수점 단위(FPU), IEEE 754 호환
- 하위 영역 및 배경 영역이 있는 8개의 MPU 영역(선택 사항)
- 통합 비트 필드 처리 명령어 및 버스 레벨 비트
- 웨이크업 인터럽트 컨트롤러
- 통합 WFI 및 WFE 지침 및 종료 시 슬립 기능, 슬립 및 딥 슬립 신호, 암 전원 관리 키트가 포함된 선택적 유지 모드
- 선택적 JTAG 및 직렬 와이어 디버그 포트. 최대 8개의 중단점 및 4개의 감시점
- 선택적 명령어 추적(ETM), 데이터 추적(DWT) 및 계측 추적(ITM)

# ARM(Advanced RISC Machine)

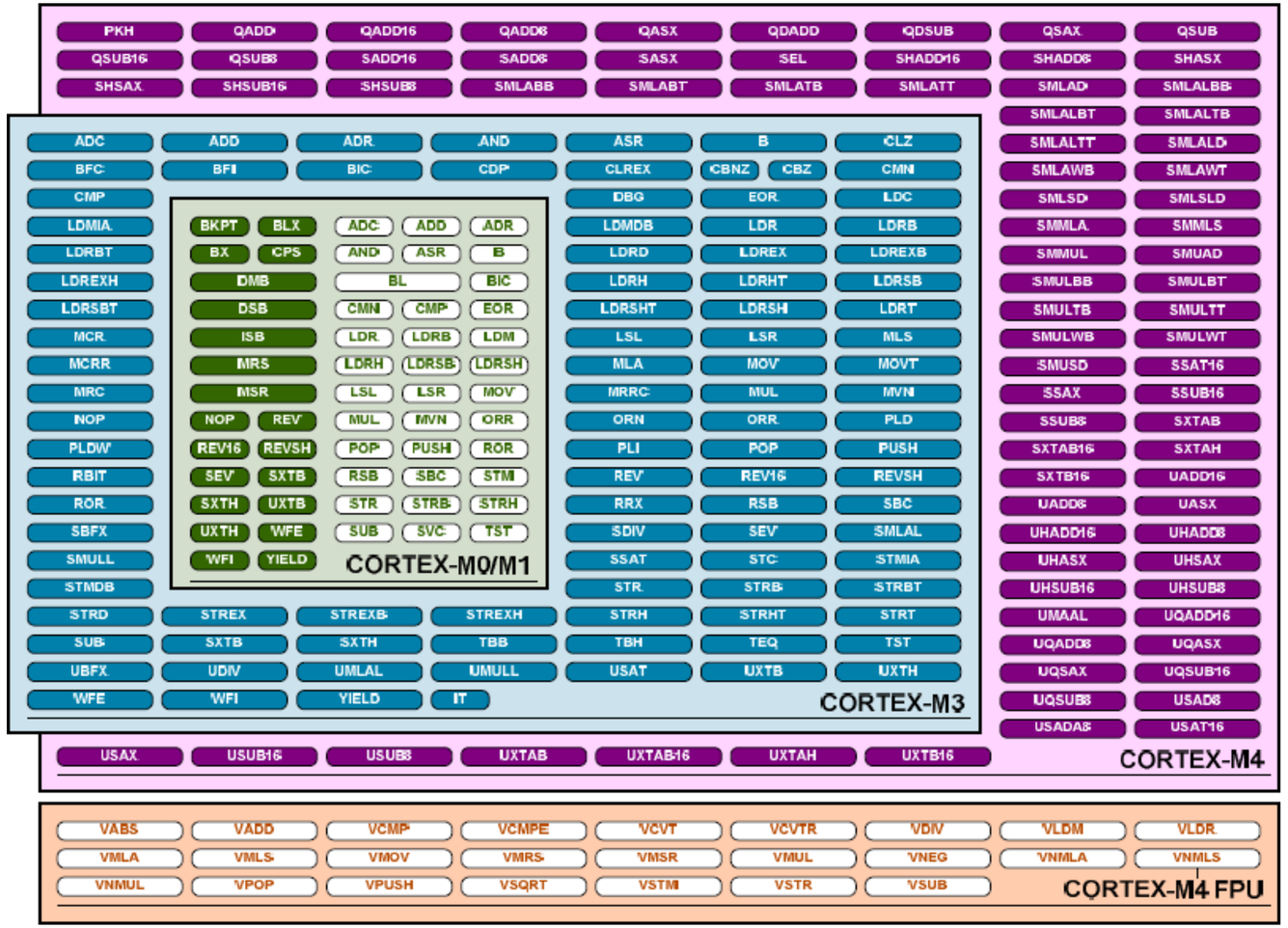


## ARM 명령어 Set










〈참조〉 별첨 : ARMCortexM4_TechnicalRefrenceManual.pdf

# ARM Assembly Programming



□ ARM32_AssemblyProgrammingGuide.pdf 의 내용을 이용하여 ARM Assembly 프로그램에 대해 알아보고 다음과 같은 조건의 Assembly 프로그램을 작성해 봅시다.

0에서 100까지의 정수를 출력하는 프로그램

- 3의 배수 인 경우 "Multiple of 3" 출력
- 5의 배수 인 경우 "Multiple of 5" 출력
- 3 과 5의 배수 인 경우 "Multiple of 3 & 5"를 출력
- 위의 항목에 해당되지 않는 경우 해당 정수 인쇄

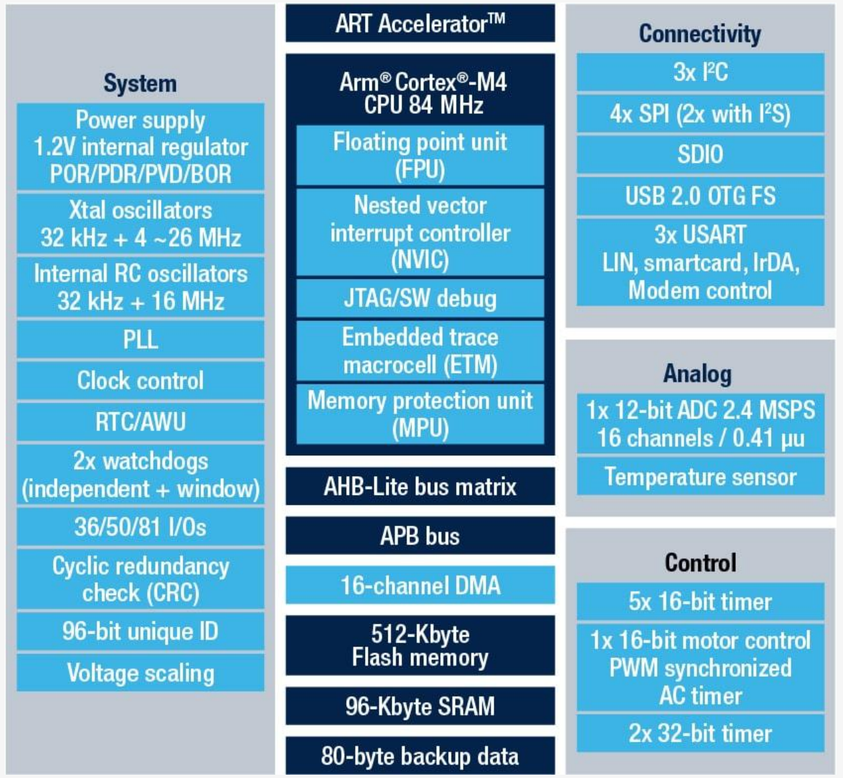
- 온라인 ARM32 Assembler : https://cpulator.01xz.net/?sys=arm-de1soc

# NUCLEO-F401RE 보드 소개



## 1. 마이크로 컨트롤러

- 프로세서
  STM32F401RET6 in LQFP64 package
  ARM® 32-bit Cortex®-M4 CPU with FPU(부동 소수점 연산기)
  84 MHz max CPU frequency
- 전원
  VDD from 1.7 V to 3.6 V
- 메모리
  512 KB Flash
  96 KB SRAM
- 기능
  외부 인터럽트가 가능한 GPIO 50개
  16채널 12-bit ADC
  RTC 지원
  Advanced-Control Timer
  General Purpose Timers 7개
  Watchdog Timers 2개
  USART/UART 4 포트
  I2C 3개
  SPI 3개
  SDIO 지원
  USB 2.0 OTG FS




〈출처〉 https://www.st.com/en/microcontrollers-microprocessors/stm32f401re.html

# NUCLEO-F401RE 보드 소개



## 2. 보드의 특징

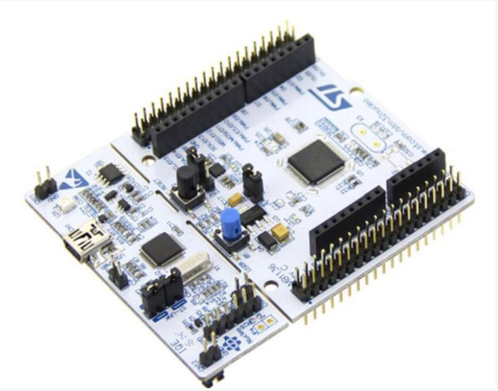
- 기능 2가지의 확장 핀
  Arduino Uno R3 호환핀 배치
  STM32 I/Os 핀을 위한 확장 핀(Morpho headers)

- ST-LINK/V2-1 debugger/programmer를 포함
  ST-LINK/V2-1 기능이 있고 다른 장치를 위한 프로그래밍 툴로서 활용 가능

- 유연한 전원 공급
  USB 전원 또는 외부전원 (3.3 V, 5 V, 7 – 12 V)

- 실험용 LED 와 푸시 버튼이 기본으로 장착

- USB를 통한 3가지의 접속방법 제공
  USB 디스크 드라이브 (Mass Storage Disk) 기능으로 파일 탐색기에서 이진 프로그램을 드래그 앤 드롭으로 업로드 가능
  가상 통신 포트 (Virtual Com port) : USB를 통하여 UART 시리얼 통신이 가능
  Debug port : 디버그 포트로 동작

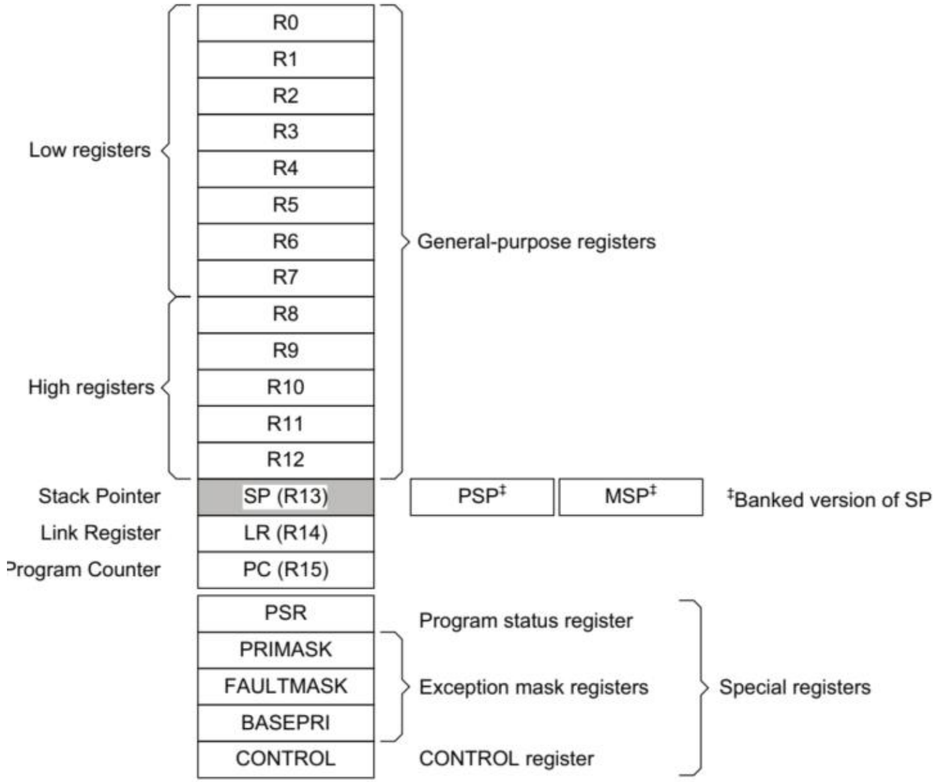


〈STmicro 사의 NUCLEO F401RE〉

# NUCLEO-F401RE 보드 소개



## □ STM32 Coretex-M4 주요 레지스터




〈그림〉 주요 레지스터

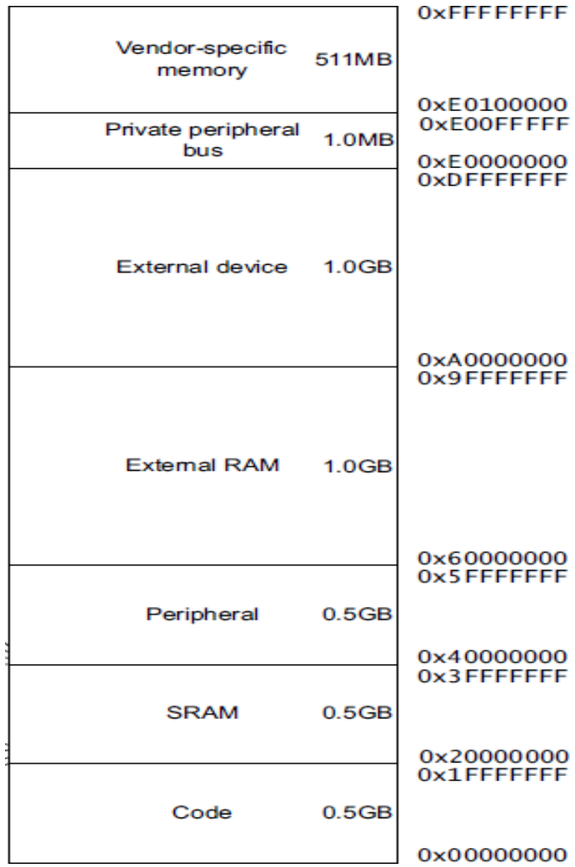
- All registers are 32 bits wide
- 13 general purpose registers
  - Registers r0 – r7 (Low registers)
  - Registers r8 – r12 (High registers)
  - Use to hold data, addresses, etc.
- 3 registers with special meaning/usage
  - Stack Pointer (SP) – r13
  - Link Register (LR) – r14
  - Program Counter (PC) – r15
- xPSR – Program Status Register
  - Composite of three PSRs
  - Includes ALU flags (N,Z,C,V)

〈참조〉 별첨 : STM32 Cortex –M4 MCUs and MPUs programming manual (en.DM00046982.pdf)

# NUCLEO-F401RE 보드 소개



## □ STM32 Coretex-M4 메모리 맵



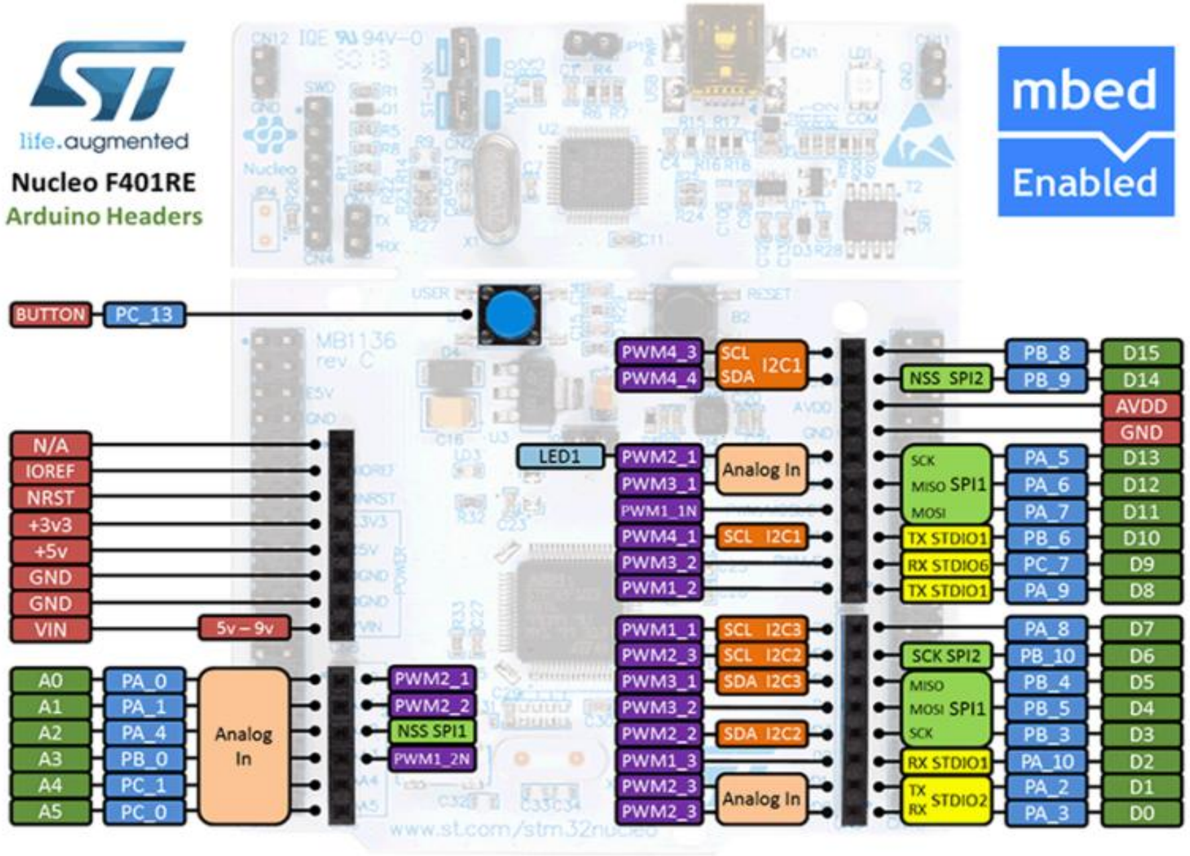

〈그림〉 메모리 맵

〈참조〉 별첨 : STM32 Cortex –M4 MCUs and MPUs programming manual (en.DM00046982.pdf)

# NUCLEO-F401RE 보드 소개



- Nucleo401RE 핀 배치도 (1 of 2)

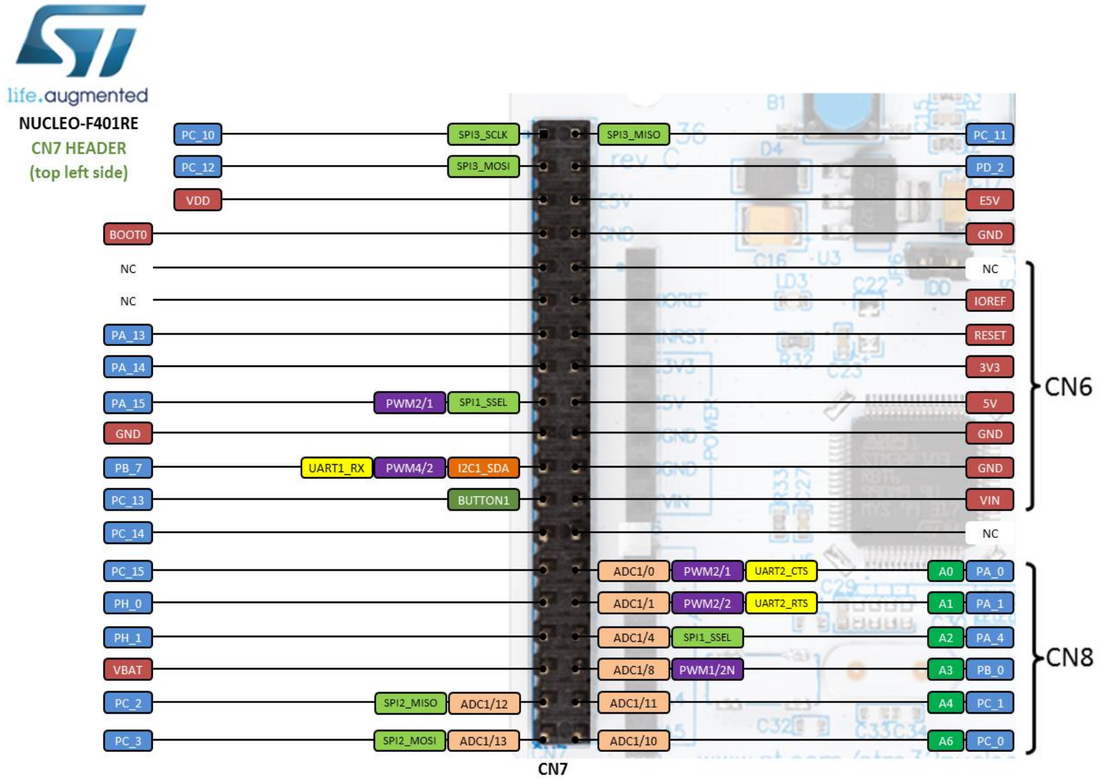


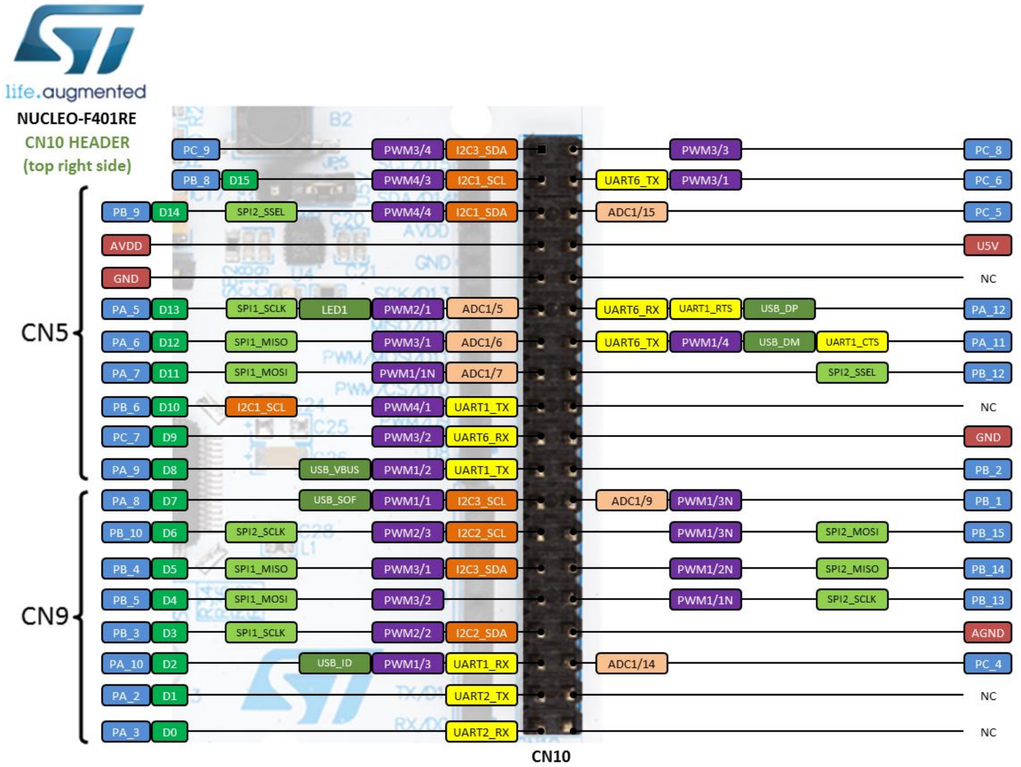

〈출처〉 https://os.mbed.com/platforms/ST-Nucleo-F401RE/

# NUCLEO-F401RE 보드 소개



## Nucleo401RE 핀 배치도 (2 of 2)




〈출처〉 https://os.mbed.com/platforms/ST-Nucleo-F401RE/

# ARM32 Development Environment



□ STM32 32-bit 마이크로 콘트롤러 – ARM Cortex-M 프로세서에 기반

□ 시스템 요구 사양

- Windows 10/Linux 64bit/macOS
- USB Type A or C to Micro-B cable / to mini-B cable / to C cable

□ 개발 툴 체인

- IAR Systems – IAR Embedded Workbench (https://www.iar.com/ewarm ) : 상용
- Keil – MDK-ARM (https://www.keil.com/ ) : 상용
- ARM Mbed Studio (https://os.mbed.com/studio /) : 무료
- STMicroelectrionics – STM32CubeIDE (https://www.st.com/en/development-tools/stm32cubeide.html ) :무료

# 개발환경 구축 (STM32CubeIDE)



## STM32CubeIDE 소개

- STM사에서 제공하는 STM32용 무료 툴로 코드의 생성, 컴파일, 디버깅까지 지원하는 통합 개발 환경 도구
- 코드 생성 툴인 STM32CubeMX와 Eclipse 기반의 GCC 툴체인 TrueSTUDIO의 결합
- STM사의 디버거인 ST-Link지원
- 윈도우, 리눅스, 맥 OS에 설치 가능한 올인원 다중 OS 개발 도구

## 설치

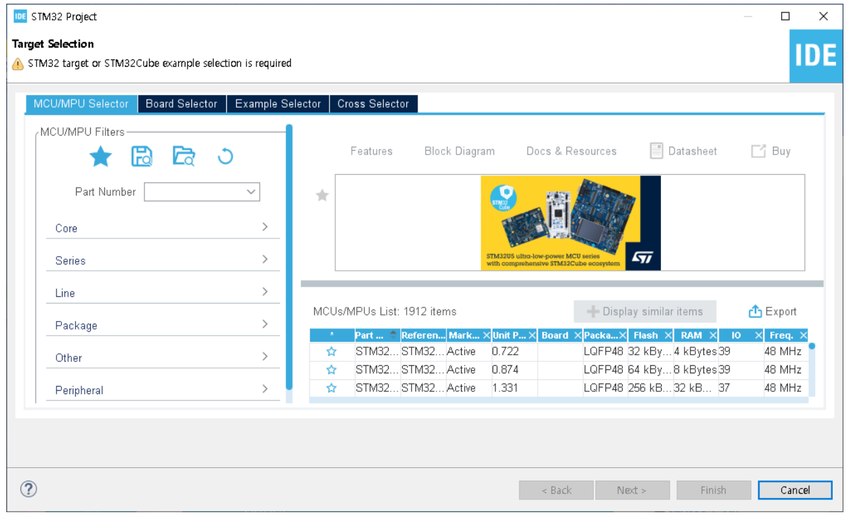
- https://www.st.com/en/development-tools/stm32cubeide.html#get-software 에서
- 적절한 OS 선택, 다운로드 후 .exe 파일 실행하여 설치
- 자바 런타임 필요 – https://www.java.com/ko/download

## 문서

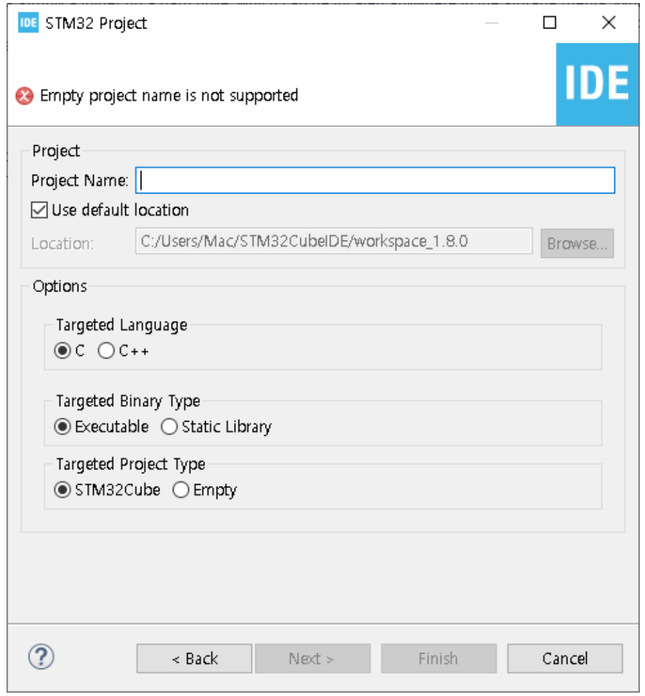
- https://www.st.com/en/development-tools/stm32cubeide.html#documentation
- https://wiki.st.com/stm32mcu/wiki/Category:STM32CubeIDE

# 프로젝트 생성

## Project 생성 (File 〉New 〉STM32 Project )

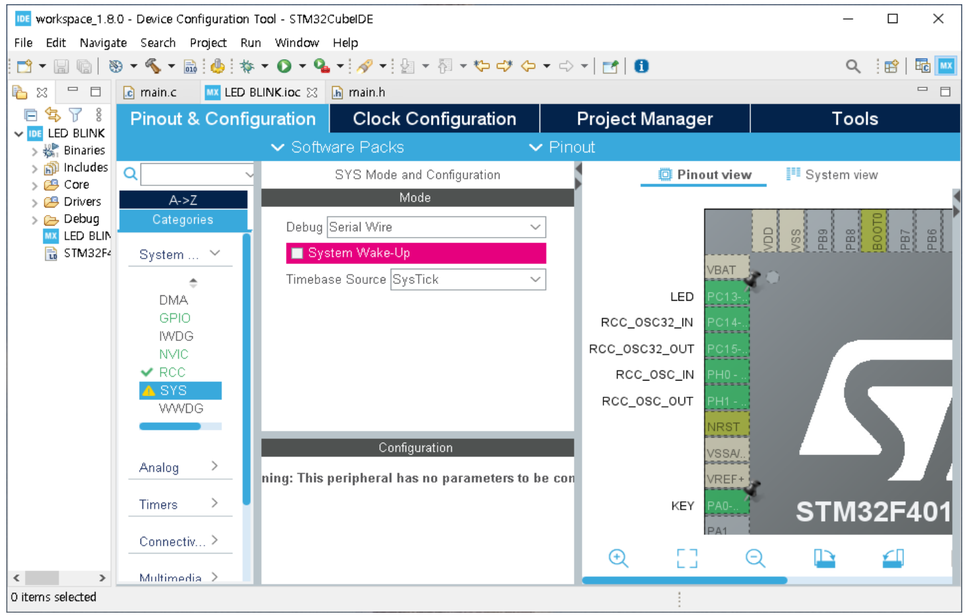



대상 MCU 또는 보드를 선택하고 다음 페이지로 이동




프로젝트명 입력 Finish

## 주변장치 설정 및 프로젝트 툴 체인 설정




주변 장치 pinout 설정
클록 구성
프로젝트 툴체인/IDE : STM32CuceIDE

## 파일 〉저장 → code 생성

# 초기화 코드



## □ 전원이 켜진 후 CPU 초기화 진행

| | |
|---|---|
| Reset 벡터 시작 | CPU는 처음 부팅 시 리셋 벡터 주소로부터 시작하여 순차적으로 코드를 실행 |
| ↓ | |
| ROM to RAM 복사 | Flash 메모리 등의 ROM에서 RAM으로 복사하는 과정 수행 |
| ↓ | |
| main 함수로 점프 | 메인 함수로 점프하여 순차로 초기화 코드를 실행 |
| ↓ | |
| 주변장치 초기화 | main 함수 이후 GPIO, Flash 메모리, 시스템 clock, UART 등의 주변장치들을 초기화 |

# 초기화 코드



## □ 초기화 startup 코드 (startup_stm32f401xc.s)

```
 .syntax unified
 .cpu cortex-m4
 .fpu softvfp
 .thumb

.global  g_pfnVectors
.global  Default_Handler

 .section  .text.Reset_Handler
 .weak  Reset_Handler
 .type  Reset_Handler, %function
Reset_Handler:
 ldr   sp, =_estack      // set stack pointer

// Copy the data segment initializers from flash to SRAM
 ldr r0, =_sdata
 ldr r1, =_edata
 ldr r2, =_sidata
 movs r3, #0
 b LoopCopyDataInit

CopyDataInit:
 ldr r4, [r2, r3]
 str r4, [r0, r3]
 adds r3, r3, #4

LoopCopyDataInit:
 adds r4, r0, r3
 cmp r4, r1
 bcc CopyDataInit
```

```
// Zero fill the bss segment.
 ldr r2, =_sbss
 ldr r4, =_ebss
 movs r3, #0
 b LoopFillZerobss

FillZerobss:
 str  r3, [r2]
 adds r2, r2, #4

LoopFillZerobss:
 cmp r2, r4
 bcc FillZerobss

// Call the clock system intitialization function.
 bl  SystemInit
// Call static constructors
   bl __libc_init_array
// Call the application's entry point.
 bl  main
 bx  lr
.size  Reset_Handler, .-Reset_Handler
```

# GPIO



## GPIO 제어

- 4개의 32비트 구성 레지스터(GPIOx_MODER, GPIOx_OTYPER, GPIOx_OSPEEDR, GPIOx_PUPDR)
- 2개의 32비트 데이터 레지스터(GPIOx_IDR 및 GPIOx_ODR)
- 1개의 32비트 세트/리셋 레지스터(GPIOx_BSRR)
- 1개의 32비트 잠금 레지스터(GPIOx_LCKR)
- 2개의 32비트 대체 기능 선택 레지스터(GPIOx_AFRH 및 GPIOx_AFRL)

rm0368-stm32f401xbc-and-stm32f401xde-advanced-armbased-32bit-mcus-stmicroelectronics.pdf 중 8. GPIO 참조

## GPIO 관련 HAL 함수

| HAL 함수<br>HAL_Init() 로 초기화 | 직접 제어 |
|---|---|
| HAL_GPIO_ReadPin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)<br>〈사용 예〉<br>HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_0) | <br><br>if(GPIOA -〉 IDR & GPIO_PIN_0) |
| HAL_GPIO_WritePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)<br>〈사용 예〉<br>HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_SET); /* HIGH 출력 */<br>HAL_GPIO_WritePin(GPIOC, GPIO_PIN_13, GPIO_PIN_RESET); /* LOW 출력 */ | <br><br>GPIOC -〉 ODR \|= GPIO_PIN_13; //(1 〈〈 13)<br>GPIOC -〉 ODR &= ~GPIO_PIN_13; //~(1 〈〈 13) |
| HAL_GPIO_TogglePin(GPIO_TypeDef * GPIOx, uint16_t GPIO_Pin)<br>〈사용 예〉<br>HAL_GPIO_TogglePin(GPIOC, GPIO_PIN_13); | <br><br>GPIOC -〉 ODR ^= GPIO_PIN_13; |

# LED Blink(1)



## GPIO 초기화

```
/* Initialize all configured peripherals */
/* USER CODE BEGIN 2 */
/* Initialize all configured peripherals */
MX_GPIO_Init();
/* USER CODE END 2 */
. . .
```

```
. . .
/* USER CODE BEGIN 4 */
static void MX_GPIO_Init(void)
{
GPIO_InitTypeDef GPIO_InitStruct = {0};

/* GPIO Ports Clock Enable */
__HAL_RCC_GPIOA_CLK_ENABLE();

/*Configure GPIO pin Output Level */
HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);

/*Configure GPIO pin : PA5 */
GPIO_InitStruct.Pin = GPIO_PIN_5;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

}
/* USER CODE END 4 */
```

```
/* Exported functions prototypes ----------------
----------------------------*/
void Error_Handler(void);


/* USER CODE BEGIN EFP */
static void MX_GPIO_Init(void);
/* USER CODE END EFP */
```

inc/main.h

# LED Blink(2)



## HAL 이용 vs 직접 레지스터 제어

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();

  while (1)
  {
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
    HAL_Delay(1000);
  }
}
```

```
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();

  while (1)
  {
    GPIOC -> ODR &= ~GPIO_PIN_5;
    HAL_Delay(1000);
    GPIOC -> ODR |= GPIO_PIN_5;
    HAL_Delay(1000);
  }
}
```

# Button



□ 앞의 예제를 참고하여 입력 장치인 Button을 제어해봅시다.

- 실습 1: 버튼을 누르면 LED가 켜지고 손을 떼면 LED가 꺼지는 프로그램

  HAL_GPIO_ReadPin ( )/ HAL_GPIO_WritePin ( ) 함수 이용

- 실습 2 : 버튼을 누르면 200mS 간격으로 LED가 blink 하는 프로그램

  HAL_Delay(BLINK_TICK1)  //200mS  함수 이용

# 인터럽트(1)



□ 인터럽트 벡터 테이블
- ISR(Interrupt Service Routine)의 시작 주소(벡터) 등록 : 4바이트로 구성
- 최대 496개의 외부 인터럽트
- 인터럽트/예외 벡터 테이블은 일반적으로 시작 코드 파일에 존재

□ **인터럽트 서비스 루틴(ISR) 처리**
- 인터럽트 신호를 수신할 때 프로세서는 다중 사이클 명령어를 제외하고 현재 수행하던 명령어 완료
- stack overflow에 주의

□ EXTI(External Interrupt)
- 외부에서 신호가 입력될 경우 Device에 이벤트 또는 인터럽트가 발생되는 기능
- 개별적으로 트리거 이벤트(Rising Edge, Falling Edge, Rising & Falling Edge)를 선택하도록 구성하거나 독립적으로 마스킹
- Priority가 높은 것부터 Active 된다. (숫자가 작을 수록 Priority가 높음)

□ EXTI 주요 기능
- 각 인터럽트/이벤트 라인에서 독립적인 트리거 및 마스크
- 각 인터럽트 라인에 대한 전용 상태 비트
- 최대 23개의 소프트웨어 이벤트/인터럽트 요청 생성
- APB2 클록 주기보다 낮은 펄스 폭으로 외부 신호 감지

# 인터럽트(2)



## 하드웨어 인터럽트 선택

- 23개의 인터럽트 라인의 마스크 비트 구성(EXTI_IMR)
- 인터럽트 라인의 트리거 선택 비트 구성(EXTI_RTSR 및 EXTI_FTSR)
- 외부 인터럽트 컨트롤러(EXTI)에 매핑된 NVIC IRQ 채널을 제어하는 활성화 및 마스크 비트를 구성하여 23개 라인 중 하나에서 오는 인터럽트를 올바르게 인식할 수 있도록 한다.

## 하드웨어 이벤트 선택

- 23개 이벤트 라인의 마스크 비트 구성(EXTI_EMR)
- 이벤트 라인의 트리거 선택 비트 구성(EXTI_RTSR 및 EXTI_FTSR)

rm0368-stm32f401xbc-and-stm32f401xde-advanced-armbased-32bit-mcus-stmicroelectronics.pdf 중 10. Interruptand events 참조

# 인터럽트 실습



## External Interrupt (EXTI) Pinout 설정

- GPIO PC13 : GPIO_EXTI 으로 선택
- PC13-WKUP Configuration 에서
  - SYSTEM CORE의 GPIO Mode : "External Interrupt Mode with Rising edge trigger detection"
  - GPIO Pull-up /Pull-down : Pull-up 선택
  - User Label ：INT0
- 동작 확인 : LED (PA5 출력 설정, Label LED 로 설정)

## callback 함수 구현 설정

- 사용자가 EXTI발생 시 구현할 동작은 Drivers/STM32F4xx_HAL_Driver/Src/stm32f4xx_hal_gpio.c 로부터 복사하여 main.c 에서 구현

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
        if(GPIO_Pin==INT0_Pin){
          HAL_GPIO_TogglePin(LED_GPIO_Port, LED_Pin);
         }
}
```

# 인터럽트 확인



- **HAL_GPIO_EXTI_GET_FLAG 매크로 :** 특정 EXTI 라인의 인터럽트 플래그를 확인

```
if(__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_0))
{
  // EXTI 라인 0에 대한 인터럽트 발생
  // 인터럽트 처리 코드

  // 플래그 클리어
  __HAL_GPIO_EXTI_CLEAR_FLAG(GPIO_PIN_0);
}
```