ARM Coretex-M

펌웨어 설계

CPU 초기화 코드 설계

한국기술교육대학교
온라인평생교육원

학습목표

- CPU 초기화 코드에 대해 설명할 수 있다.
- STM32F429의 초기화 코드를 생성·분석·설계할 수 있다.

학습내용

- CPU 초기화 코드 소개
- STM32F429 초기화 코드 설계

CPU 초기화 코드 소개

개요

CPU 초기화 코드에 대한 이해

CPU 초기화 코드

처음 전원이 들어온 후 CPU가 본연의 기능을 하기 위해
초기화해야 하는 블록들을 초기 설정하는 코드

CPU의 초기화 코드를 이해하기 위해
CPU의 내부 하드웨어 구조와 어셈블리어를 알아야 함

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정

초기화 코드 작성 ▶ Reset 벡터 시작 ▶ 데이터의 복사 : ROM → RAM ▶ Main 함수로 점프 ▶ 주변장치 초기화

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정

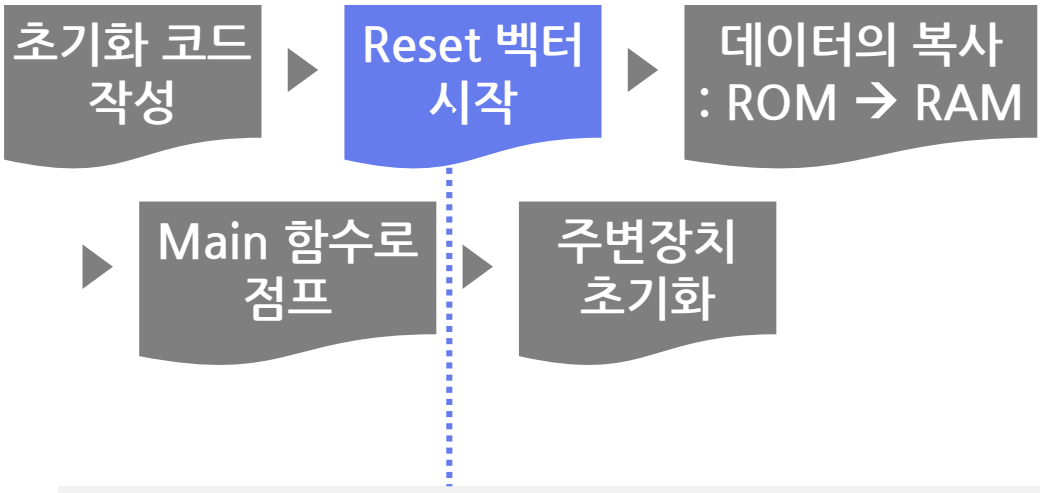
초기화 코드 작성 ▶ Reset 벡터 시작 ▶ 데이터의 복사 : ROM → RAM ▶ Main 함수로 점프 ▶ 주변장치 초기화

- 초기화 코드는 대부분 어셈블리어로 작성됨
  - C 언어 등의 고 수준 언어에 필요한 stack heap 등은 메모리 초기 설정 이후에 동작하기 때문
- 초기화 코드에서 제일 처음 초기화하는 하드웨어 블록은 메모리와 CPU의 clock과 같은 기본적인 블록

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정



CPU는 처음 부팅 시 Reset 벡터라고 하는 어드레스로 시작해서 순차적으로 코드를 실행

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정

초기화 코드 작성 ▶ Reset 벡터 시작 ▶ 데이터의 복사 : ROM → RAM ▶ Main 함수로 점프 ▶ 주변장치 초기화

⋯› 일반적으로 데이터를 Flash 메모리 등의 ROM에서 RAM으로 복사하는 과정 진행

- RAM의 stack과 heap 등 기능별 영역으로 나누어 초기화함
- 이 외 라이브러리 함수를 위한 RAM 영역을 초기화

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정

초기화 코드 작성 ▶ Reset 벡터 시작 ▶ 데이터의 복사 : ROM → RAM ▶ Main 함수로 점프 ▶ 주변장치 초기화

CPU 초기화 코드 소개

개요

CPU 초기화 코드의 일반적인 과정

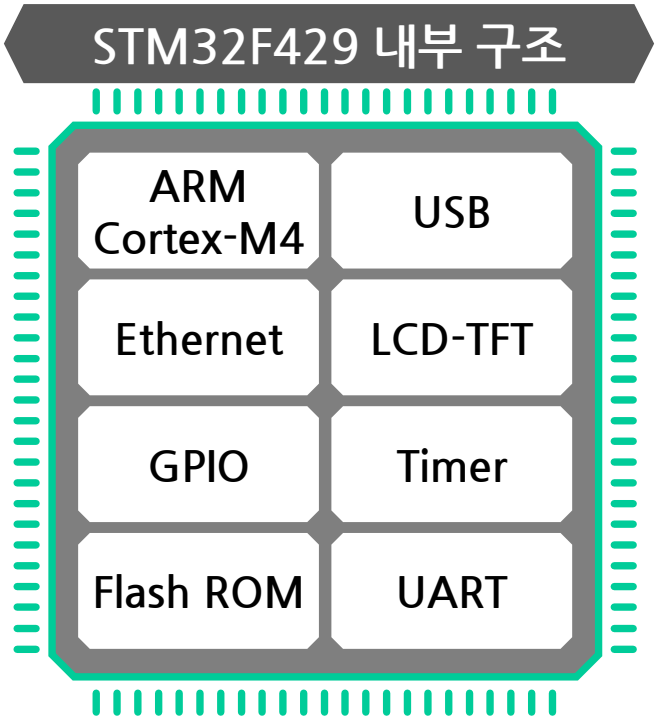
초기화 코드 작성 ▶ Reset 벡터 시작 ▶ 데이터의 복사 : ROM → RAM ▶ Main 함수로 점프 ▶ 주변장치 초기화

⋯› Main 함수 이후 GPIO, Flash 메모리, 시스템 clock, UART 등의 주변장치들을 초기화함

CPU 초기화 코드 소개

개요

STM32F429 MCU



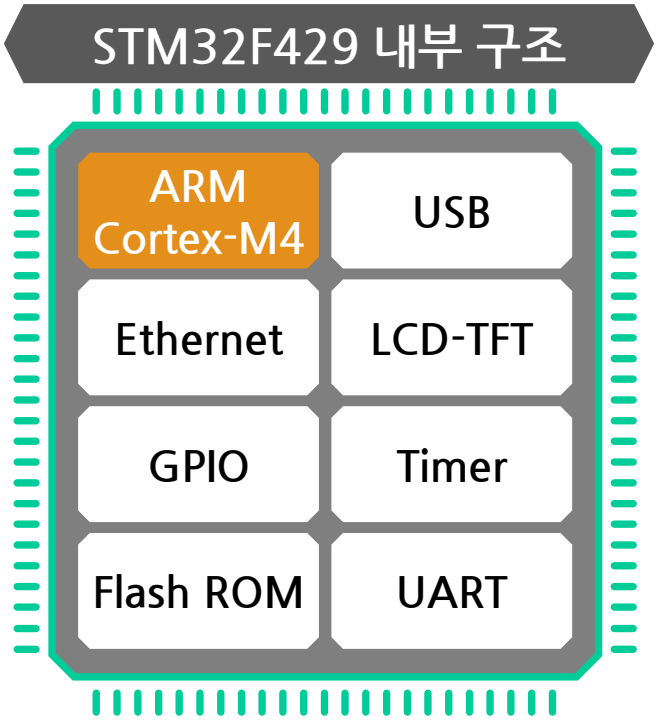
초기화 코드는
주로 ARM Cortex-M4의 내부, 메모리, 타이머 등
가장 기본이 되는 블록들을 초기화함

초기화 코드를 이해하기 위해
기본 내부 블록에 대해 살펴봐야 함

CPU 초기화 코드 소개

개요

STM32F429 MCU



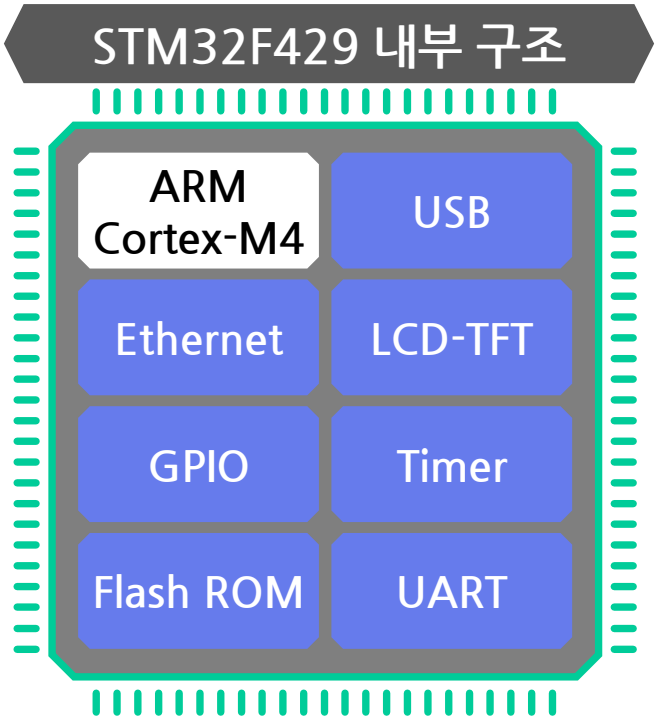
ARM Cortex-M4

⋯› STM32F429 MCU의 Core임

CPU 초기화 코드 소개

개요

STM32F429 MCU



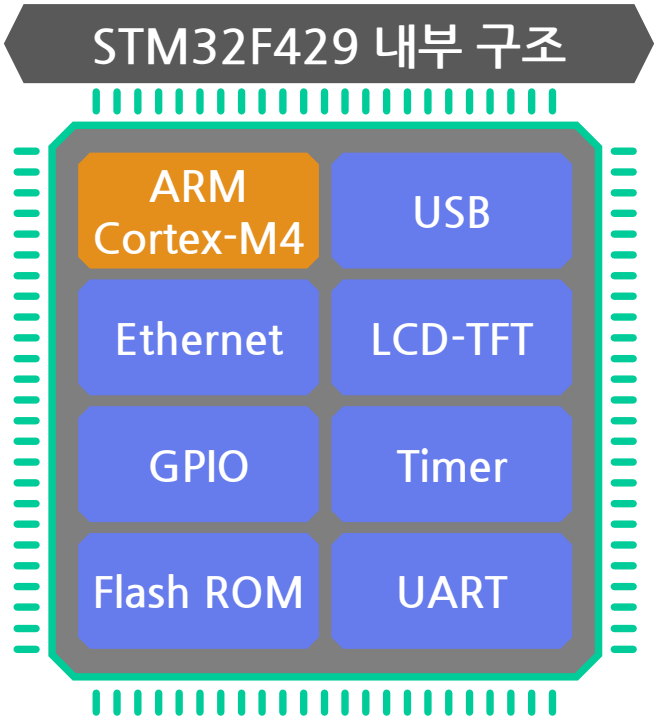
IP

- USB, Ethernet 등의 하드웨어 블록을 IP라고 함
- Intellectual Property의 약자임

CPU 초기화 코드 소개

개요

STM32F429 MCU



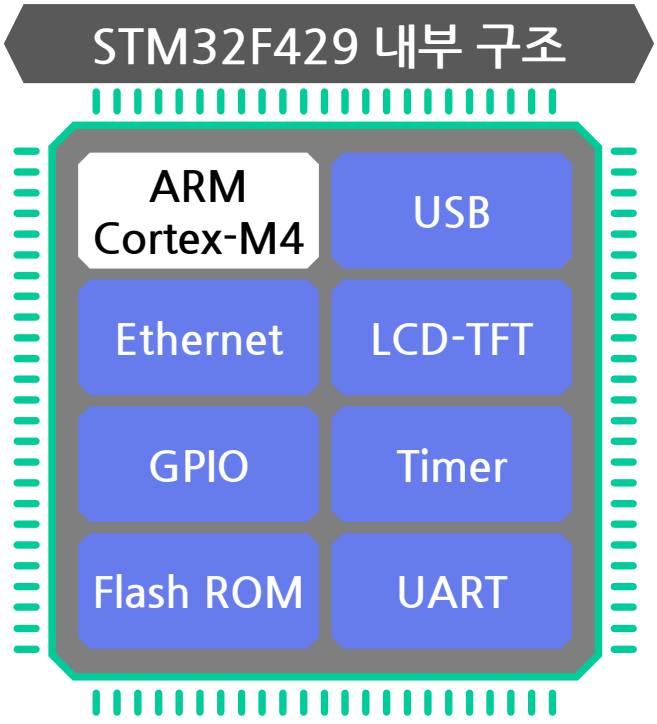
ARM 사의 Cortex-M4 구매 + 통신 블록, clock 관련 블록, 메모리 블록, 미디어 블록 등의 IP 추가

STM 사에서의 'STM32F429' MCU 완성

CPU 초기화 코드 소개

개요

STM32F429 MCU



통신 블록, clock 관련 블록,
메모리 블록,
미디어 블록 등의 IP 추가

↓

SoC

⋯› Core 외 필요한 IP들을 추가한 칩을 말함

⋯› System on Chip의 약자로,
Chip 하나에 여러 기능이 동작하는 시스템
을 구축해 놓았다는 의미임

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor

- FPU
  - 소수점 연산기 있는 Cortex-M4 core
- NVIC
  - 인터럽트 컨트롤러
- 디버깅 관련 장치
  - FPB, DWT, ITM 등
- 메모리 보호 블록
  - MPU
- AHB, AHB Access Port, Bus Matrix
  - APB와 같은 버스를 연결하기 위한 장치

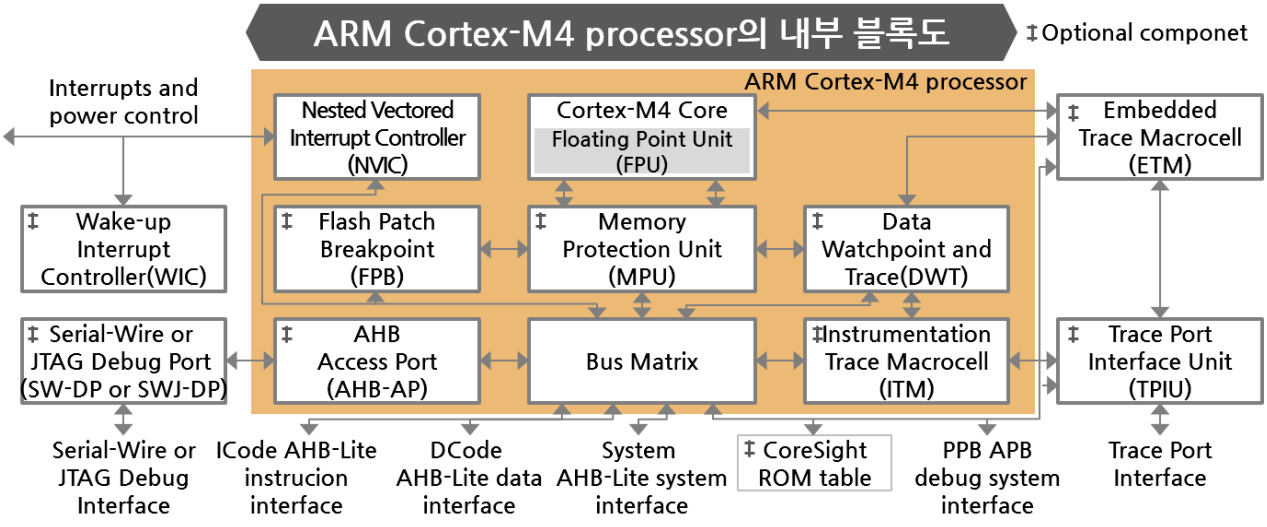
CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor

- Cortex-M4 processor 와 USB, Ethernet과 같은 컨트롤러들을 연결하기 위해서 AHB와 APB, 버스를 통해 연결이 가능함.
  - AHB와 APB는 ARM 사에서 정의한 AMBA 버스라는 규격 중 하나임
    : AHB는 빠른 속도의 데이터 전송이 필요한 블록을 연결할 때, APB는 느린 속도의 블록을 연결할 때 사용됨
  - 버스란 여러 가닥의 Address와 여러 가닥의 Data 라인으로 이루어진 통신 포트임

- STM 사에서 Cortex-M4 processor에 GPIO 컨트롤러와 같은 저속의 디바이스는 자사에서 설계한 GPIO 컨트롤러를 APB 버스에 연결하여 제어함



CPU 초기화 코드 소개

개요

- ARM Cortex-M4 processor Core 레지스터

> 어셈블리어는 Core 레지스터를 피연산자로 동작하므로,
> 어셈블리어를 이해하기 위해
> Core 내부의 General purpose 레지스터에 대해 알아야 함

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor Core 레지스터

내부 레지스터들(이름이 R로 시작하며 크기가 32비트임)이 존재

R0	…	R12	R13(SP)	R14(LR)	R15(PC)	xPSR

총 13개의 범용 레지스터로,
일반적인 데이터 연산을 위해 사용

R0	…	R12	R13(SP)	R14(LR)	R15(PC)	xPSR

stack pointer 레지스터로,
SP_process, SP_main이라는 banked 레지스터를 말함

SP_main	SP_process
MSP라고도 하며, OS 커널, exception handler에서 사용	PSP라고도 하며, 사용자 어플리케이션 코드에서 사용

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor Core 레지스터

- 내부 레지스터들(이름이 R로 시작하며 크기가 32비트임)이 존재

R0	…	R12	R13(SP)	R14(LR)	R15(PC)	xPSR

Link 레지스터로,
함수 호출 시 되돌아올 주소를 저장

R0	…	R12	R13(SP)	R14(LR)	R15(PC)	xPSR

Program Counter로,
현재 실행하는 명령어의 위치를 나타냄

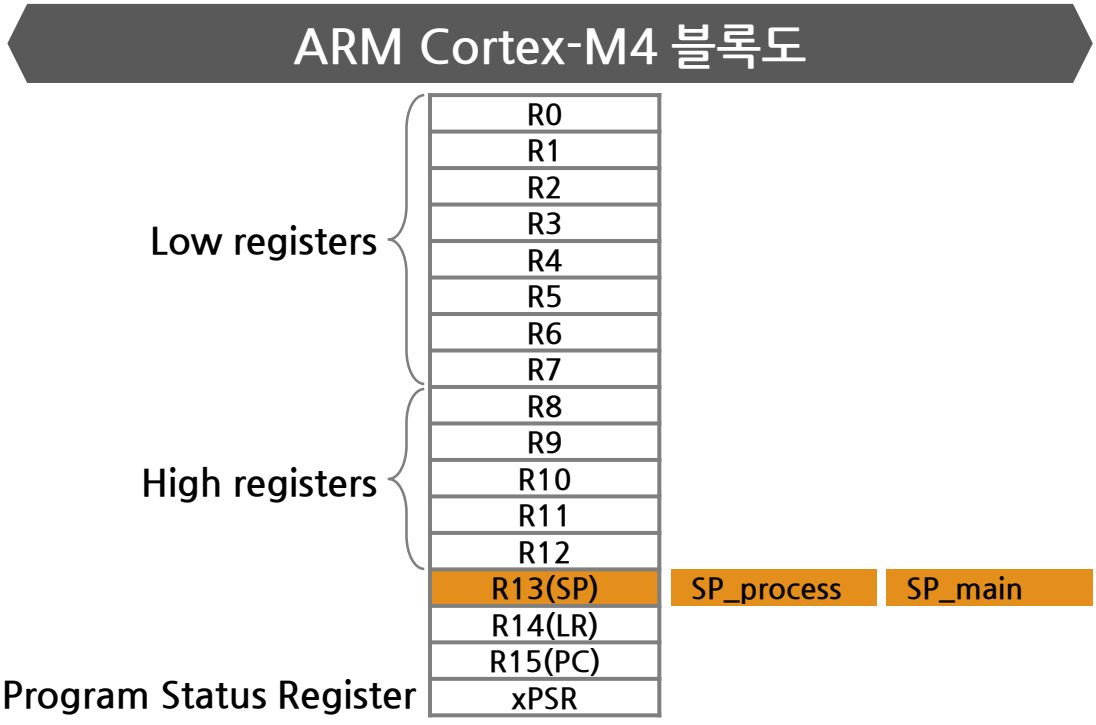
R0	…	R12	R13(SP)	R14(LR)	R15(PC)	xPSR

'Special-purpose Program Status Register'라는 xPSR로
명령어 실행 후 결과 등을 저장

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor Core 레지스터



CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor의 동작 모드와 특권 레벨

잘못된 명령어의 실행으로 인해 프로세서 전체가
오동작하는 것을 막기 위해 사용자 프로그램과
OS 커널 실행을 구분하기 위한 다음 모드와 레벨을 제공

동작 모드	특권 레벨
Thread mode	Privileged level
Handler mode	Unprivileged level

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor의 동작 모드와 특권 레벨

동작 모드

Thread mode

- 어플리케이션 프로그램 실행 시 사용되며, 프로세서가 Reset이 되면 이 모드가 됨
- System Control Block 같은 중요한 정보의 접근이 근본적으로 차단됨

Handler mode

- OS 커널, Exception 핸들러 등이 실행할 때 사용되는 모드
- System Control Block의 접근 가능

CPU 초기화 코드 소개

개요

ARM Cortex-M4 processor의 동작 모드와 특권 레벨

특권 레벨

Privileged level

- 모든 명령어와 모든 메모리 범주에 접근이 가능함

Unprivileged level

- MSR, MRS 같은 일부 명령어는 접근이 불가함
- System timer, NVIC, SCB 등의 접근이 제한됨

CPU 초기화 코드 소개

개요

ARM Cortex-M4 어셈블리어

초기화 코드는 주로 어셈블리어로 되어 있기 때문에
초기화 코드를 설계하기 위해
ARM Cortex-M4 core용 어셈블리어를 알아야 함

- 어셈블리어를 다른 용어로 Instruction set라고도 함
- ARM Cortex-M4는 32비트 Core이기 때문에
  32비트 명령어로 이루어짐
  → 어셈블리어를 컴파일한 최종 기계어는
  32비트 숫자 값으로 이루어짐

```
Reset_Handler:
  ldr   sp, =_estack      상수값
/*
Core 레지스터
movs r1, Core 레지스터
  b  LoopCopyDataInit
Instruction
CopyDataInit:
  ldr  r3, =_sidata       상수값
  ldr  r3, Core 레지스터
  str  r3, [r0, r1]
Instruction
```

조기화 ARM 어셈블리어 코드

CPU 초기화 코드 소개

개요

ARM Cortex-M4 어셈블리어

```
Reset_Handler:
  ldr   sp, =_estack      /* set stack pointer
*/

movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4
```

초기화 ARM 어셈블리어 코드

ldr sp, =_estack

- 한 번에 실행되는 32비트 명령어
- sp인 stack pointer 레지스터에 estack이라는 상수값을 저장한다는 의미의 instruction

CPU 초기화 코드 소개

개요

ARM Cortex-M4 어셈블리어

```
Reset_Handler:
  ldr   sp, =_estack      /* set stack pointer
*/

movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4
```

초기화 ARM 어셈블리어 코드

Reset_Handler:

- 레이블로, C언어의 함수명과 같은 개념임
- 다른 위치에서 점프할 때 위치를 지정하기 위해 사용

CPU 초기화 코드 소개

개요

ARM Cortex-M4 어셈블리어

```
Reset_Handler:
  ldr   sp, =_estack      /* set stack pointer
*/

movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4
```

초기화 ARM 어셈블리어 코드

b LoopCopyDataInit

- b는 branch 명령어로, LoopCopyDataInit이라는 레이블로 점프하라는 명령어임

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

- CubeMX를 사용하여 생성한 프로젝트에서 초기화를 담당하는 파일은 'startup_stm32f429xx.s'임
  - 파일 확장자 's' : GNU계열의 어셈블리 파일에 주로 사용되는 확장자

- CPU는 부팅을 하면 제일 먼저 reset vector로 점프하여 reset handler를 수행함
- 초기화 파일의 위치는 프로젝트 폴더의 startup₩startup_stm32f429xx.s임

Startup_stm32f429xx.s의 주요 동작

stack pointer 초기화	program counter를 Reset Handler로 설정	Exception 벡터에 vector table entry를 설정	Main 함수로 점프

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
    .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 */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
```

초기화 어셈블리어 파일

Reset_Handler

- CPU가 부팅 후 제일 먼저 실행하는 코드
- 제일 먼저 ldr sp, =_estack을 통해 stack pointer를 초기화

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
/* Copy the data segment initializers from
flash to SRAM */
  movs  r1, #0
  b  LoopCopyDataInit

CopyDataInit:
  ldr  r3, =_sidata
  ldr  r3, [r3, r1]
  str  r3, [r0, r1]
  adds  r1, r1, #4

LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, = edata
```

초기화 어셈블리어 파일

CopyDataInit 함수

- Flash 메모리에 저장된 값들을 data 영역의 메모리로 복사하여 초기화

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
LoopCopyDataInit:
  ldr  r0, =_sdata
  ldr  r3, =_edata
  adds  r2, r0, r1
  cmp  r2, r3
  bcc  CopyDataInit
  ldr  r2, =_sbss
  b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
  movs  r3, #0
  str  r3, [r2], #4

LoopFillZerobss:
```

초기화 어셈블리어 파일

LoopCopyDataInit

⋯› data 영역과 bss 영역을 초기화 하는 함수

data 영역	bss 영역
특정한 초기값 있는 변수 (C언어의 전역변수, static 변수 등)	전역변수, 즉 초기값이 0으로 지정되는 변수들을 모아 놓음

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  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
```

초기화 어셈블리어 파일

LoopFillZerobss 함수

- bss 영역의 메모리를 0으로 초기화

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  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
```

초기화 어셈블리어 파일

SystemInit 함수

- clock 설정, 인터럽트 벡터 설정, 외부 메모리 설정, FPU 설정 등을 수행
- src/system_stm32f4xx.c에 위치

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

초기화 파일

```
LoopFillZerobss:
  ldr  r3, = _ebss
  cmp  r2, r3
  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
```

초기화 어셈블리어 파일

bl main

··› C언어의 시작점인 main 함수로 점프

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

초기화 코드 startup_stm32f429xx.s를 이해하기 위해
Linker script에 대해 알아야 함

- 컴파일 후 링크 시 사용하는 값들을 정리한 스크립트 파일을 말함
  - 컴파일 : 소스 코드를 기계어로 변환된 object 파일로 만드는 것
  - 링크 : 여러 object 파일들을 합쳐 하나의 출력파일로 만드는 과정

- STM32F429ZITx_FLASH.ld 파일이 Linker script 파일
- 가장 기본적인 RAM, ROM의 시작주소와 크기, reset vector의 위치, stack과 heap의 크기 등이 지정되어 있음
- 초기화 파일 startup_stm32f429xx.s의 많은 상수들이 Linker script 파일에서 지정한 값들을 사용

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20030000;    /* end of RAM */
/* Generate a link error if heap and stack
don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required
amount of heap  */
_Min_Stack_Size = 0x400; /* required amount
of stack */

/* Specify the memory areas */
MEMORY
{
```

Linker script 파일

ENTRY

- 프로그램이 처음 실행되는 진입점을 지정할 때 사용
- 'Reset_Handler'이라는 심볼명이 이 프로그램의 진입점이라는 의미임
- startup_stm32f429xx.s에 Reset_Handler 함수가 존재

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20030000;    /* end of RAM */
/* Generate a link error if heap and stack
don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required
amount of heap  */
_Min_Stack_Size = 0x400; /* required amount
of stack */

/* Specify the memory areas */
MEMORY
{
```

Linker script 파일

_estack = 0x2003000

- stack 주소로, RAM의 끝 주소를 나타냄
- CPU가 제일 처음 실행하는 Reset_Handler에서 stack pointer를 이 값으로 설정하는 것을 알 수 있음

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x20030000;    /* end of RAM */
/* Generate a link error if heap and stack
don't fit into RAM */
_Min_Heap_Size = 0x200;      /* required
amount of heap  */
_Min_Stack_Size = 0x400; /* required amount
of stack */

/* Specify the memory areas */
MEMORY
{
```

Linker script 파일

_Min_Heap_Size, _Min_Stack_Size

- heap과 stack의 최소 크기를 지정

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

```
/* Specify the memory areas */
MEMORY
{
RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 192K
CCMRAM (rw) : ORIGIN = 0x10000000, LENGTH = 64K
FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 2048K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH
  .isr_vector :
  {
    . = ALIGN(4);
```

Linker script 파일

MEMORY

⋯› RAM와 ROM의 영역을 지정

RAM	ROM
시작주소는 0x20000000, 크기는 192KB로 지정	Flash 메모리의 시작 주소는 0x80000000, 크기는 2MB로 지정

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Linker Script

```
RAM (xrw) : ORIGIN = 0x20000000, LENGTH = 192K
CCMRAM (rw) : ORIGIN = 0x10000000, LENGTH = 64K
FLASH (rx) : ORIGIN = 0x8000000, LENGTH = 2048K
}

/* Define output sections */
SECTIONS
{
  /* The startup code goes first into FLASH */
  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector)) /* Startup code */
    . = ALIGN(4);
  } >FLASH
```

Linker script 파일

SECTIONS

- 메모리의 영역을 특성별로 쪼갠 것
- 링크할 때 특성이 같은 데이터와 코드를 모아 출력 실행파일을 만들 때 사용

예 .isr_vector :
인터럽트 핸들러들만 모아둔 영역

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Main.c

- 생성된 C 코드의 시작점임
- Main 함수는 src/main.c에 존재함

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Main.c

```
/* MCU Configuration--------------------------*/

/* Reset of all peripherals, Initializes the
Flash interface and the Systick. */
HAL_Init();

/* USER CODE BEGIN Init */

/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */
```

Linker script 파일

HAL_Init 함수

- main 함수에서 처음 만나는 함수
- HAL는 Hardware Abstraction Layer의 약자임
- 하드웨어를 초기화하는 기능들이 포함됨
  : 여러 주변장치, Flash 메모리 인터페이스, System Tick Timer를 초기화하는 기능

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Main.c

```
  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART3_UART_Init();
  /* USER CODE BEGIN 2 */

  /* USER CODE END 2 */

  /* Infinite loop */
```

Linker script 파일

SystemClock_Config 함수

- System Clock을 설정하는 함수
  - System Clock : CPU 내부의 CPU와 AHB, APB 등 주요 버스에 관련된 clock
- 항상 구동되는 Timer인 System Tick Timer를 설정해주는 기능도 수행

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Main.c

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

Linker script 파일

MX_GPIO_Init 함수

- GPIO를 초기화하는 함수
- CubeMX에서 설정한 GPIO 관련 설정이 C 코드 형태로 포함됨
- MX로 시작하는 함수는 CubeMX로 만들어준 함수라는 의미

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Main.c

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART3_UART_Init();
/* USER CODE BEGIN 2 */

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */

  /* USER CODE BEGIN 3 */
}
```

Linker script 파일

무한 while문

- CPU가 계속 수행해야 하는 기능을 여기에 추가
- 향후 수많은 기능들을 이 코드 위치에 추가하게 됨

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Stm32f4xx_it.c

- 인터럽트 핸들러들이 모여 있는 파일
- Src 디렉토리에 위치
- CubeMX로 특정 인터럽트를 활성화하면,
  이 파일에 활성화된 인터럽트의 핸들러가 자동으로 생성됨

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Stm32f4xx_it.c

```
/**
  * @brief This function handles Non maskable in
    interrupt.
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
```

인터럽트 핸들러들

NMI_Handler

- NMI 인터럽트 핸들러
  - NMI : 'Non maskable interrupt' 약자
  - Mask 할 수 없는 인터럽트, 발생하지 못하게 할 수 없다는 의미임

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 분석

Stm32f4xx_it.c

```
/**
  * @brief This function handles Non maskable in
    interrupt.
  */
void NMI_Handler(void)
{
  /* USER CODE BEGIN NonMaskableInt_IRQn 0 */

  /* USER CODE END NonMaskableInt_IRQn 0 */
  /* USER CODE BEGIN NonMaskableInt_IRQn 1 */

  /* USER CODE END NonMaskableInt_IRQn 1 */
}

/**
```

인터럽트 핸들러들

NMI_Handler

- Cortex M 코어의 고유 인터럽트 중 하나
- CPU 제조사에서 만든 인터럽트가 아닌, ARM사에서 기본으로 제공되는 인터럽트
- 특정한 기능을 원한다면 USER CODE에 필요한 코드를 추가하면 됨

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
58
59  /* Private user code ---------------------------
60      /* USER CODE BEGIN 0 */
61
62      /* USER CODE END 0 */
63
64  /**
65    * @brief  The application entry point.
66    * @retval int
67    */
68  int main(void)
69  {
70    /* USER CODE BEGIN 1 */
71
72    /* USER CODE END 1 */
73
74
75    /* MCU Configuration--------------------------
76
77    /* Reset of all peripherals, Initializes the
78    HAL_Init();
```

/* USER CODE BEGIN 0 */
~
/* USER CODE END 0 */

자체 작성한 함수를 선언 및 정의

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
58
59  /* Private user code ---------------------------
60  /* USER CODE BEGIN 0 */
61
62  /* USER CODE END 0 */
63
64 /**
65    * @brief  The application entry point.
66    * @retval int
67    */
68   main
69 {
70     /* USER CODE BEGIN 1 */
71
72     /* USER CODE END 1 */
73
74
75    /* MCU Configuration----------------------------
76
77    /* Reset of all peripherals, Initializes the
78    HAL Init();
```

/* USER CODE BEGIN 1 */
~
/* USER CODE END 1 */

main 함수의 시작이기 때문에
대개 지역변수 등을 선언

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
77    /* Reset of all peripherals, Initializes the
78  HAL_Init()
79
80    /* USER CODE BEGIN Init */
81
82    /* USER CODE END Init */
83
84    /* Configure the system clock */
85    SystemClock_Config();
86
87    /* USER CODE BEGIN SysInit */
88
89    /* USER CODE END SysInit */
90
91    /* Initialize all configured peripherals */
92    MX_GPIO_Init();
93    MX_USART3_UART_Init();
94    /* USER CODE BEGIN 2 */
95
96    /* USER CODE END 2 */
97
```

/* USER CODE BEGIN Init */
~
/* USER CODE END Init */

HAL_Init()이 실행되고 난 다음,
추가하고자 하는 코드를 입력

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
77    /* Reset of all peripherals, Initializes the
78    HAL_Init();
79
80    /* USER CODE BEGIN Init */
81
82    /* USER CODE END Init */
83
84    /* Configure the system clock */
85    System Clock_Config()
86
87    /* USER CODE BEGIN SysInit */
88
89    /* USER CODE END SysInit */
90
91    /* Initialize all configured peripherals */
92    MX_GPIO_Init();
93    MX_USART3_UART_Init();
94    /* USER CODE BEGIN 2 */
95
96    /* USER CODE END 2 */
97
```

/* USER CODE BEGIN SysInit */
~
/* USER CODE END SysInit */

System Clock이 초기화된 후
구동할 코드를 추가

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
77    /* Reset of all peripherals, Initializes the
78    HAL_Init();
79
80    /* USER CODE BEGIN Init */
81
82    /* USER CODE END Init */
83
84    /* Configure the system clock */
85    SystemClock_Config();
86
87    /* USER CODE BEGIN SysInit */
88
89    /* USER CODE END SysInit */
90
91    /* Initialize all configured peripherals */
92    MX_GPIO_Init();
93    MX_USART3_UART_Init();
94    /* USER CODE BEGIN 2*/
95
96    /* USER CODE END 2*/
97
```

/* USER CODE BEGIN 2*/
~
/* USER CODE END 2*/

GPIO 설정이 끝나고 무한 루프에 들어가기 전, 필요한 코드를 추가

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

```
87    /* USER CODE BEGIN SysInit */
88
89    /* USER CODE END SysInit */
90
91    /* Initialize all configured peripherals */
92    MX_GPIO_Init();
93    MX_USART3_UART_Init();
94    /* USER CODE BEGIN 2 */
95
96    /* USER CODE END 2 */
97
98    /* Infinite loop */
99    /* USER CODE BEGIN WHILE */
100   while (1)
101   {
102     /* USER CODE END WHILE */
103
104       /* USER CODE BEGIN 3*/
105
106       /* USER CODE END 3*/
107   }
```

/* USER CODE BEGIN 3*/
~
/* USER CODE END 3*/

무한루프로 사용할 코드를 추가

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

- CubeMX로 생성된 main.c 파일은 개발자가 자신이 작성하고자 하는 코드를 추가할 수 있는 위치를 지정해 둠
- 주석 중 'USER CODE BEGIN'과 'USER CODE END'으로 시작하는 문장들 사이에 코드를 추가하면 됨
- 반드시 지켜야 하는 건 아니지만, 보다 깔끔한 코드를 작성할 수 있음

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

Main.c 파일

예

```
23
24  /* Private includes ----------------------
25  /* USER CODE BEGIN Includes */
26  #include "myHeader.h"
27  /* USER CODE END Includes */
28
```

▼

/* USER CODE BEGIN Includes */와
/* USER CODE END Includes */ 사이에
자신이 작성한 헤더파일들을 Include할 수 있음

예

```
44  /* Private variables ---------------------
45  UART_HandleTypeDef huart3;
46
47  /* USER CODE BEGIN PV */
48  int myVariable;
49  /* USER CODE END PV */
```

▼

PV는 Private Variables의 약자로,
/* USER CODE BEGIN PV */와
/* USER CODE END PV */ 사이에
개인이 정의한 전역 변수들을 추가 가능

STM32F429 초기화 코드 설계

STM32F429 초기화 코드 작성

stm32f4xx_it.c 파일

- 인터럽트 핸들러들이 모여 있는 파일
- 활성화된 인터럽트 핸들러에 사용자 코드를 추가하거나 수정할 수 있음

예

```
182  void SysTick_Handler(void)
183  {
184    /* USER CODE BEGIN SysTick_IRQn 0 */
185
186    /* USER CODE END SysTick_IRQn 0 */
187    HAL_IncTick();
188    /* USER CODE BEGIN SysTick_IRQn 1 */
189
190    /* USER CODE END SysTick_IRQn 1 */
```

← SysTick 인터럽트 핸들러

▼

/* USER CODE BEGIN SysTick_IRQn 1 */와
/* USER CODE END SysTick_IRQn 1 */ 사이에는
SysTick 인터럽트가 수행될 때 필요한 사용자 코드를 추가

요점노트

1. CPU 초기화 코드 소개

- CPU의 초기화 코드
  - CPU가 처음 전원이 들어온 후 CPU가 본연의 기능을 하기 위해 초기화해야 하는 블록들을 초기 설정하는 코드
  - 주로 어셈블리어로 되어 있어, 초기화 코드를 설계하기 위해 ARM Cortex-M4 core용 어셈블리어를 알아야 함
  - 초기화 코드에서 제일 처음 초기화하는 하드웨어 블록은 메모리와, CPU의 clock과 같은 기본적인 블록임
  - CubeMX 툴을 사용하여 초기화 코드를 편리하게 자동 생성할 수 있음

요점노트

2. STM32F429 초기화 코드 설계

- **STM32F429 초기화 코드 설계**
  - CubeMX를 사용하여 생성한 프로젝트에서 초기화를 담당하는 파일은 startup_stm32f429xx.s
  - CPU는 부팅을 시작하면 제일 먼저 reset vector로 점프하여 이른바 reset handler를 수행
  - Linker script는 컴파일 후 링크 시 사용하는 값들을 정리한 스크립트 파일
  - Main.c 파일에서 자체 작성한 코드를 추가하려면 USER CODE BEGIN과 USER CODE END 사이의 위치에 추가