

---

---

# 펌웨어 구조 설계

— Hancheol Cho —

---

---

# 스파게티 코드 ?



“스파게티 코드는 컴퓨터 프로그램의 소스 코드가 복잡하게 얽힌 모습을 스파게티의 면발에 비유한 표현이다. 스파게티 코드는 작동은 정상적으로 하지만, 사람이 코드를 읽으면서 그 코드의 작동을 파악하기는 어렵다.”

출처

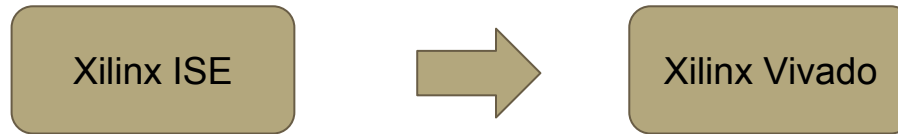
동작하는 코드를 만드는 것은  
쉽다.

제대로 만드는것은 어렵다.

# 소프트웨어 수명

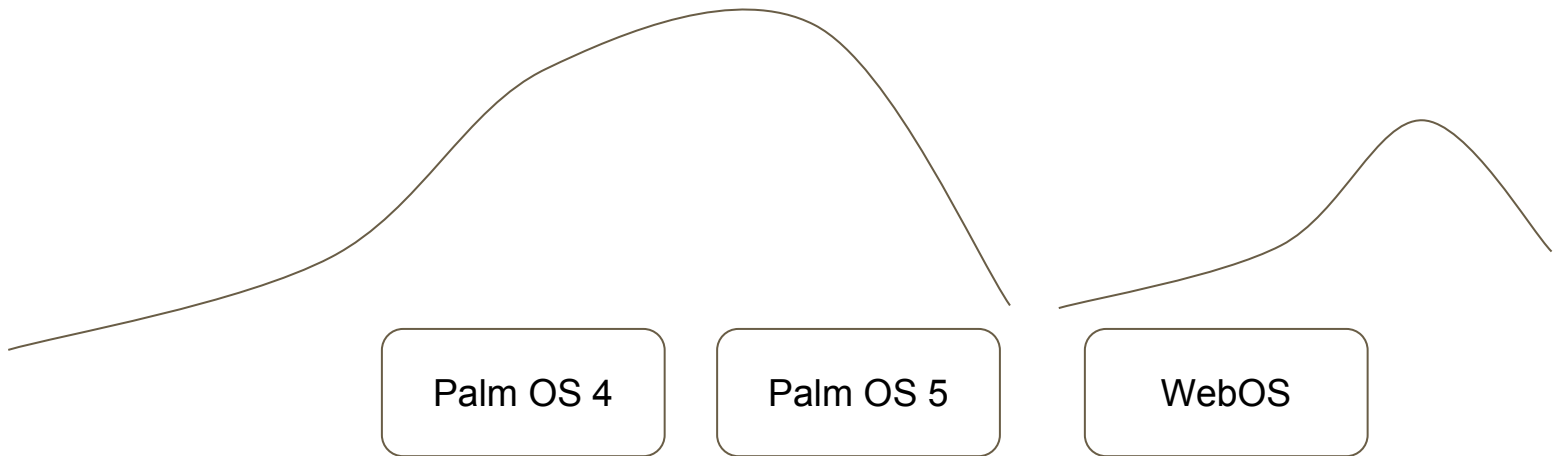
- Xilinx Vivado

“Replacing the 15 year old ISE with Vivado Design Suite took 1000 person-years and cost US \$200 million”



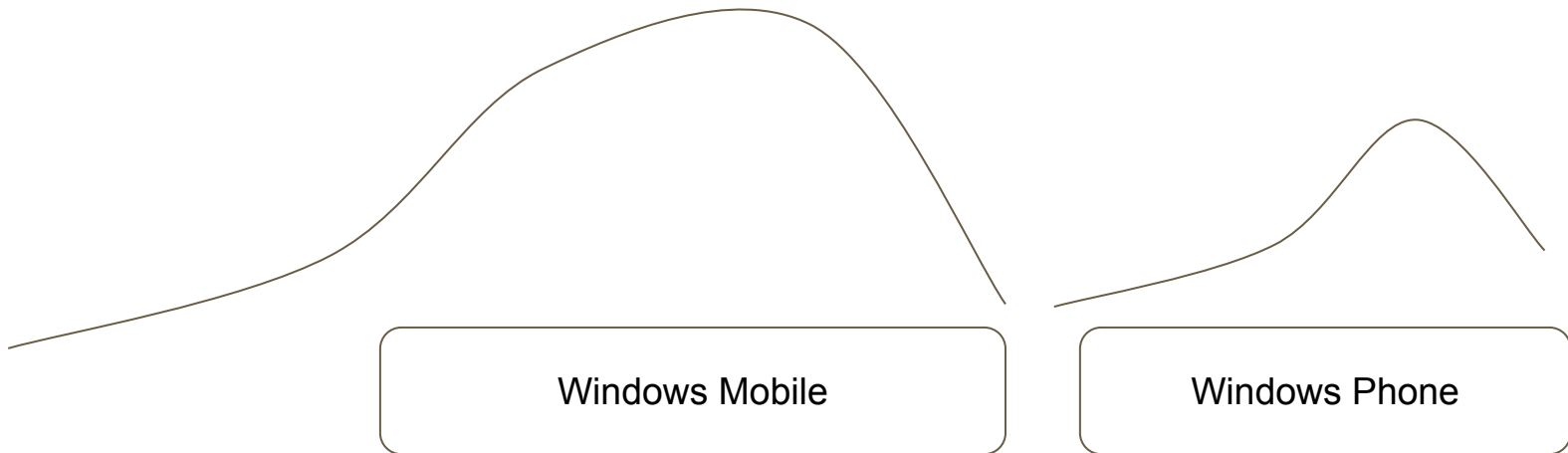
# 소프트웨어 수명

- Palm OS



# 소프트웨어 수명

- Windows Phone OS



# 소프트웨어 고려사항

재사용성

유지보수성

모듈화

사용 편의성





# 임베디드 환경이라면?

- 임베디드 환경에서는 하드웨어 제약이 많음

메모리

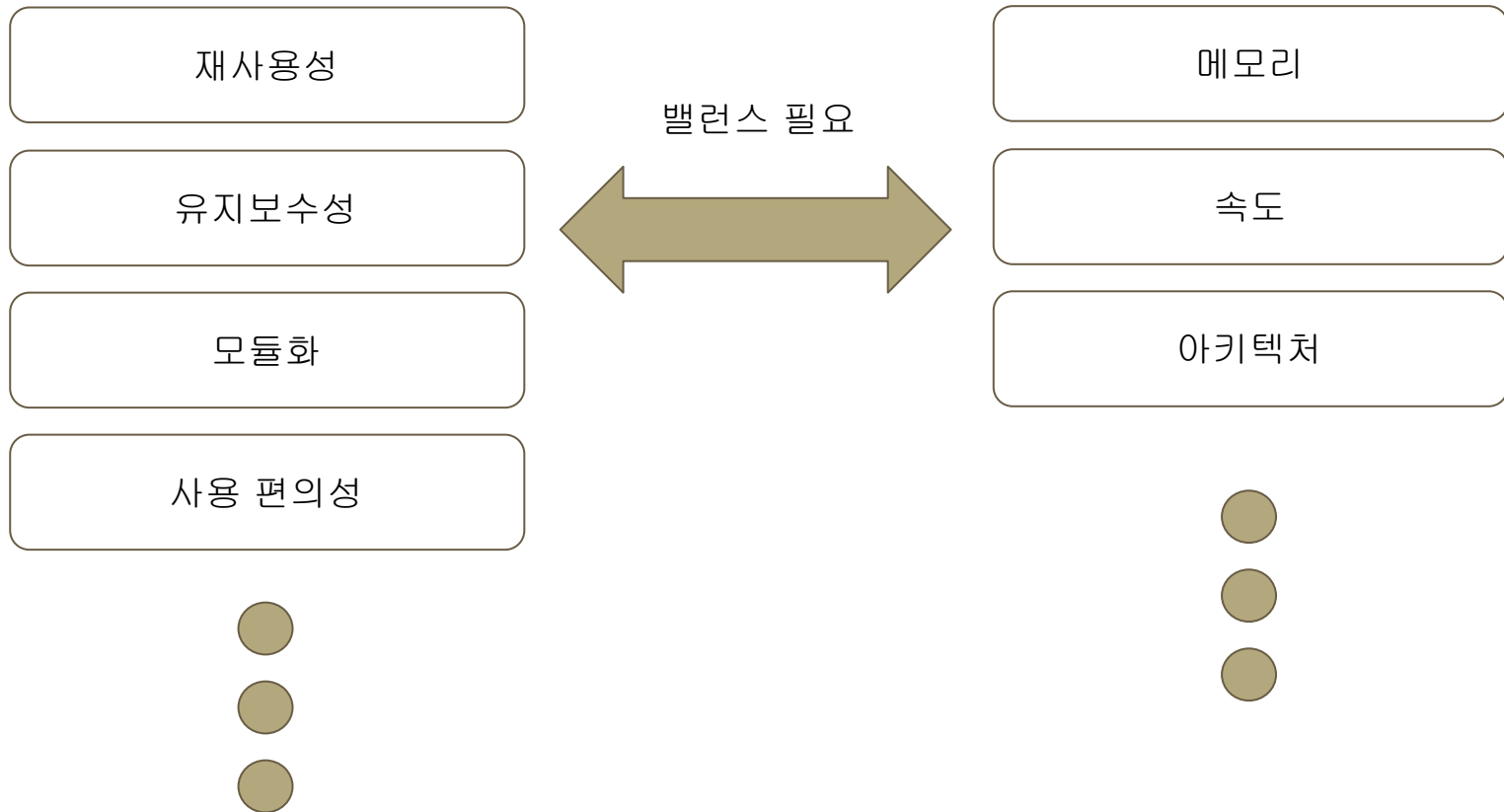
속도

아키텍처



# 펌웨어 고려 사항

- 임베디드라는 제약 사항을 고려하여 펌웨어 작성 필요



# 펌웨어 고려 사항

## 어떻게 밸런스를 맞추지?

재사용성도 유지하면서 속도도 빨라야 하고 ..

모듈화를 많이 하면서 메모리 사용량도 최소화 하고..

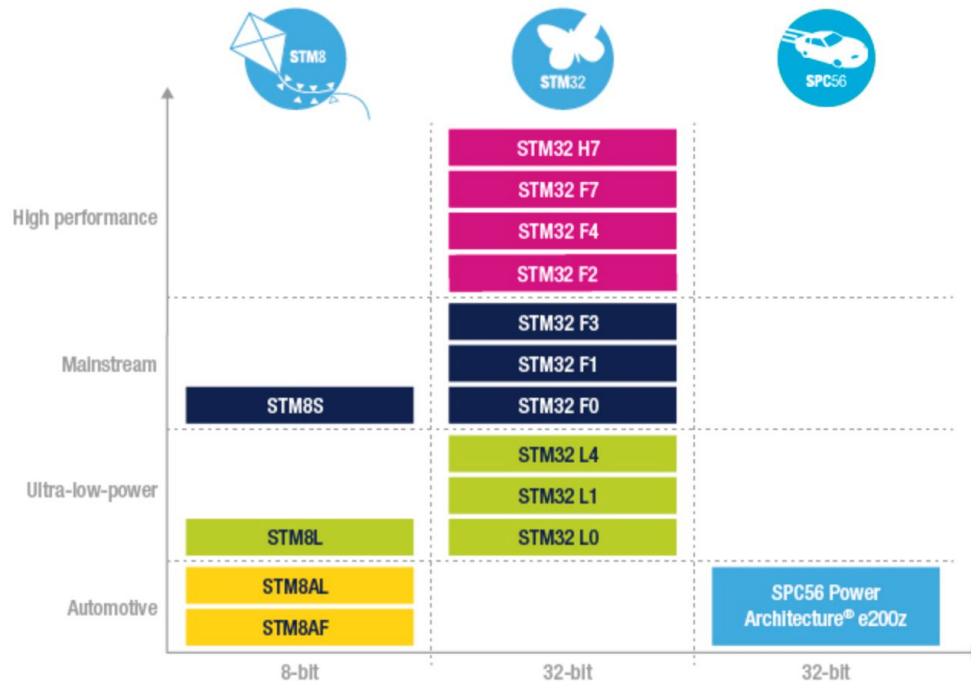


# 펌웨어 고려 사항



# 펌웨어 고려 사항

- 하드웨어가 다양함
- 각 하드웨어에 대한 깊은 이해를 하고 하드웨어에 최적화된 방법을 검토함














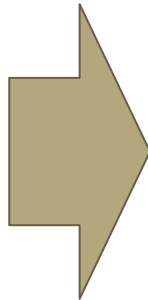
# 펌웨어 고려 사항

1. 하드웨어 특성을 잘 알고
2. 경험을 쌓고
3. 지속적인 고민을 하고
4. 공부를 게을리 하지 않는다.

# 펌웨어 고려 사항

- 칩 제조사의 어플리케이션 노트를 활용

Application Note : (32) for STM32F7 Series		
	Resource Title	
	<a href="#">AN4803</a> High-speed SI simulations using IBIS and board-level simu...	1
	<a href="#">AN4229</a> How to implement a vocoder solution using STM32 microc...	1
	<a href="#">AN1709</a> EMC design guide for ST microcontrollers	2
	<a href="#">AN4230</a> STM32 microcontrollers random number generation valida...	2
	<a href="#">AN4850</a> STM32 MCUs spread-spectrum clock generation principles...	1
	<a href="#">AN4861</a> LCD-TFT display controller (LTDC) on STM32 MCUs	2
	<a href="#">AN4286</a> - SPI 自举程序中使用的I2C 协议	4
	SPI protocol used in the STM32 bootloader	
	<a href="#">AN4946</a> Migration of microcontroller applications between STM32F...	1
	<a href="#">AN2606</a> STM32 microcontroller system memory boot mode	3
	<a href="#">AN4839</a> Level 1 cache on STM32F7 Series	1



- 칩활용에 대한 다양한 예제
- 다른 칩에도 활용 가능한 기술이 많음

# 펌웨어 고려 사항

- Errata Sheet를 반드시 확인
  - Revision에 따른 차이가 있을경우 회로 뿐만아니라 펌웨어도 각 리비전별로 호환성등에 대한 대책이 있어야함

The legend for [Table 4](#) is as follows:

A = workaround available,

N = no workaround available,

P = partial workaround available,

‘-’ and grayed = fixed.

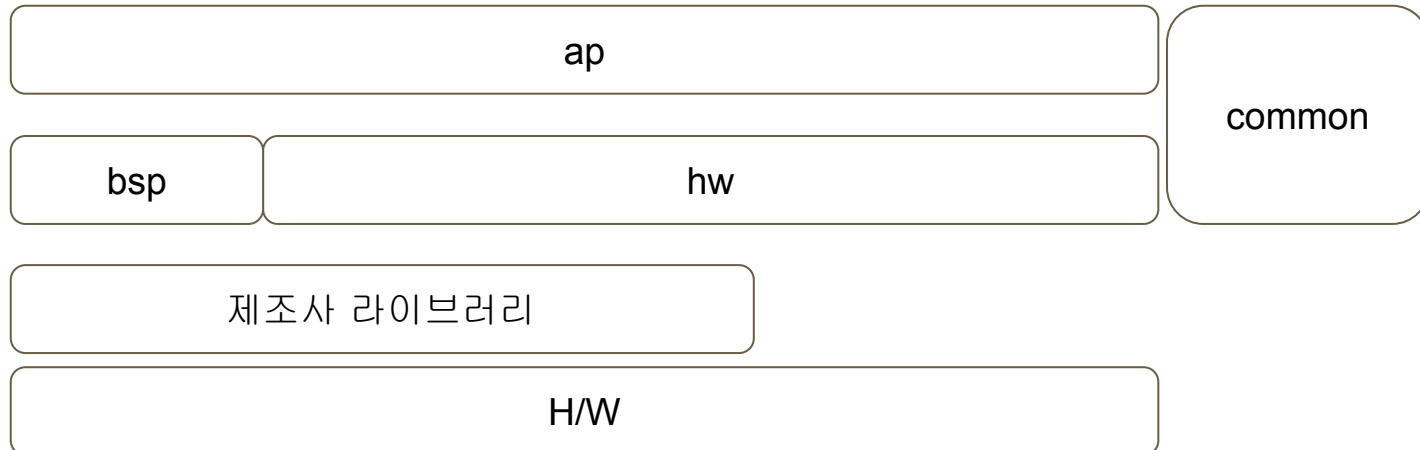
**Table 4. Summary of silicon limitations**

Links to silicon limitations		Revision A	Revision Z
Section 2.1: System limitations	<a href="#">Section 2.1.1: Internal noise impacting the ADC accuracy</a>	A	A
	<a href="#">Section 2.1.2: Wakeup from Standby mode when the back-up SRAM regulator is enabled</a>	A	A
	<a href="#">Section 2.1.3: LSE high driving and low driving capability is not usable for TFBGA216 package under certain conditions</a>	A	-
	<a href="#">Section 2.1.4: DTCM-RAM not accessible in read when the MCU is in Sleep mode (WFI/WFE)</a>	A	-



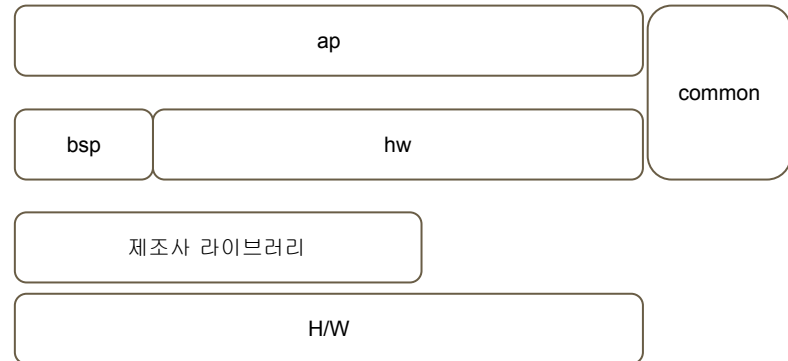
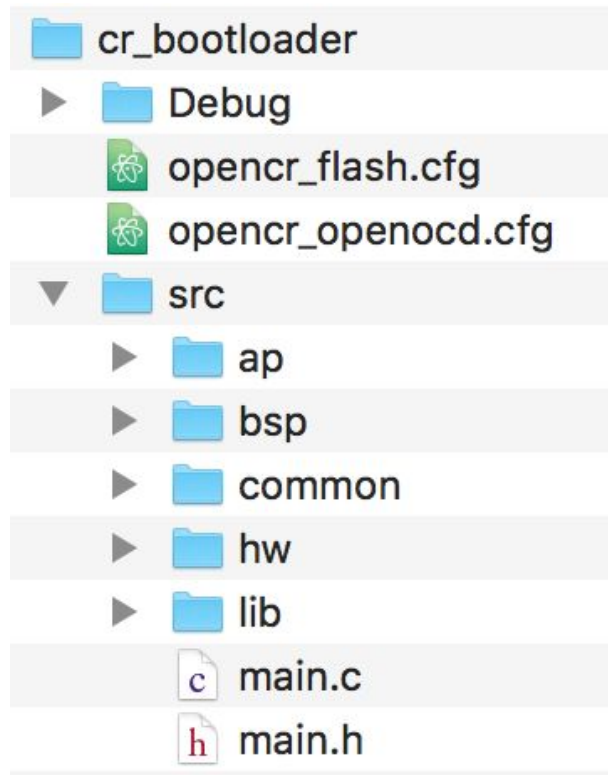
# 펌웨어 구조 설계

- 펌웨어 전체 구조를 설계
  - H/W에서 부터 상위로 Layer를 만드는 계층 구조로 설계
  - Layer간 인터페이스 범위를 설정
  - Layer수는 상황에 맞게 적절히 설정



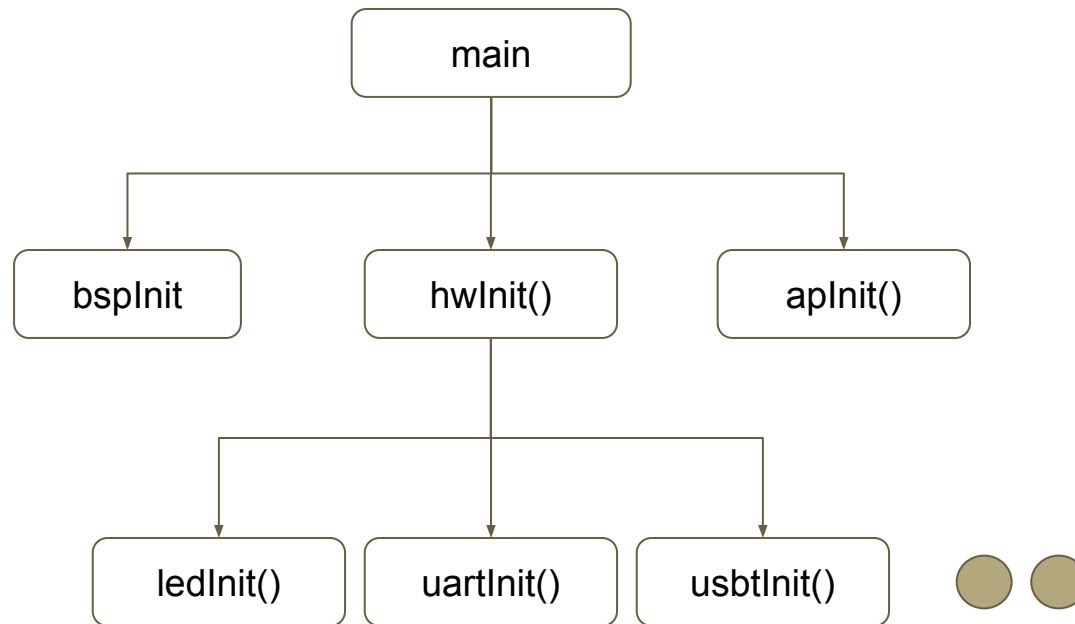
# 펌웨어 구조 설계

- 폴더 구조도 설계한 구조를 반영하도록 구성함



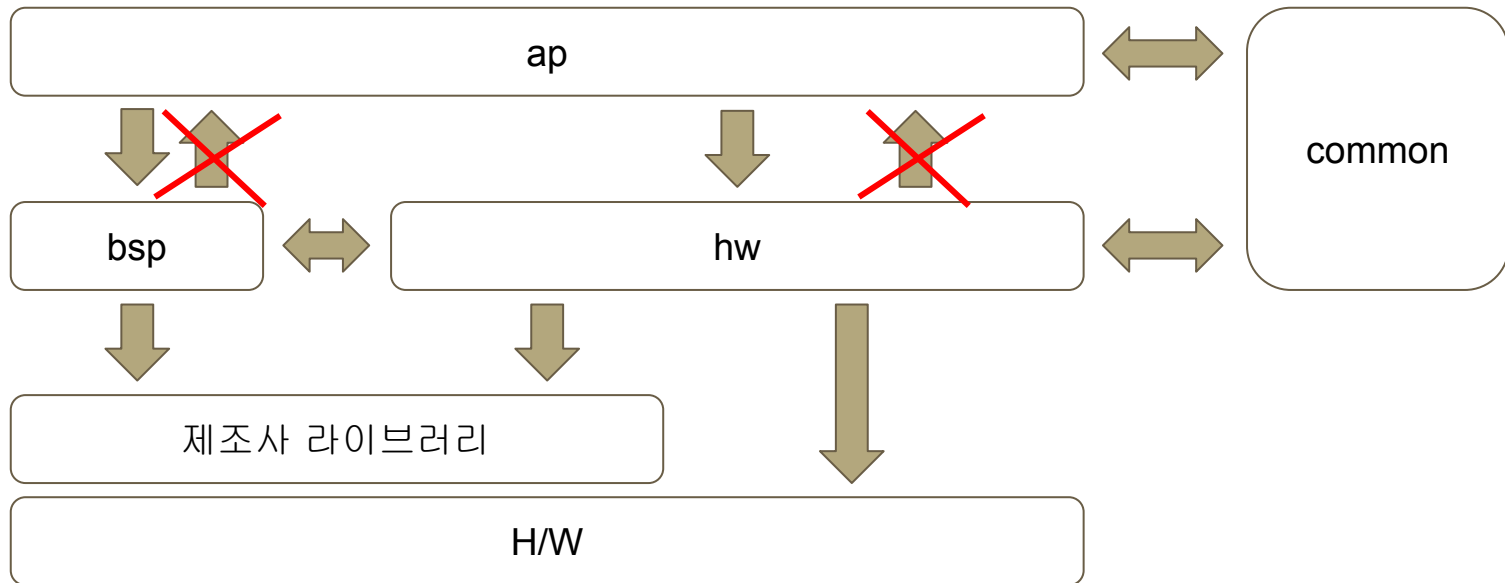
# 펌웨어 구조 설계

- 초기화 함수
  - 모듈별 초기화 함수를 기본으로 만들고 호출함



# 펌웨어 구조 설계

- 함수 호출 구조
  - 모듈 및 Layer간 함수 호출 관계를 일관되게 유지하여 의존성을 낮춤



# 모듈화 설계

- 인터페이스 함수
  - 하나의 모듈을 .h/.c로 구성하고 .h에 다른 모듈과 인터페이스를 위한 함수들을 선언함
  - 내부에 사용할 함수는 static으로 .c에 정의 함으로써 다른 모듈에서 사용하지 못하도록 함

## uart.h

```
bool uartInit(void);

uint32_t    uartOpen(uint8_t channel, uint32_t baud);
uint32_t    uartAvailable(uint8_t channel);
void        uartWaitForEnable(uint8_t channel, uint32_t timeout);
void        uartPutch(uint8_t channel, uint8_t ch);
uint8_t     uartGetch(uint8_t channel);
int32_t     uartWrite(uint8_t channel, uint8_t *p_data, uint32_t length);
uint8_t     uartRead(uint8_t channel);
int32_t     uartPrintf(uint8_t channel, const char *fmt, ...);
int32_t     uartPrint(uint8_t channel, uint8_t *p_str);
```

} 다른 모듈과  
인터페이스

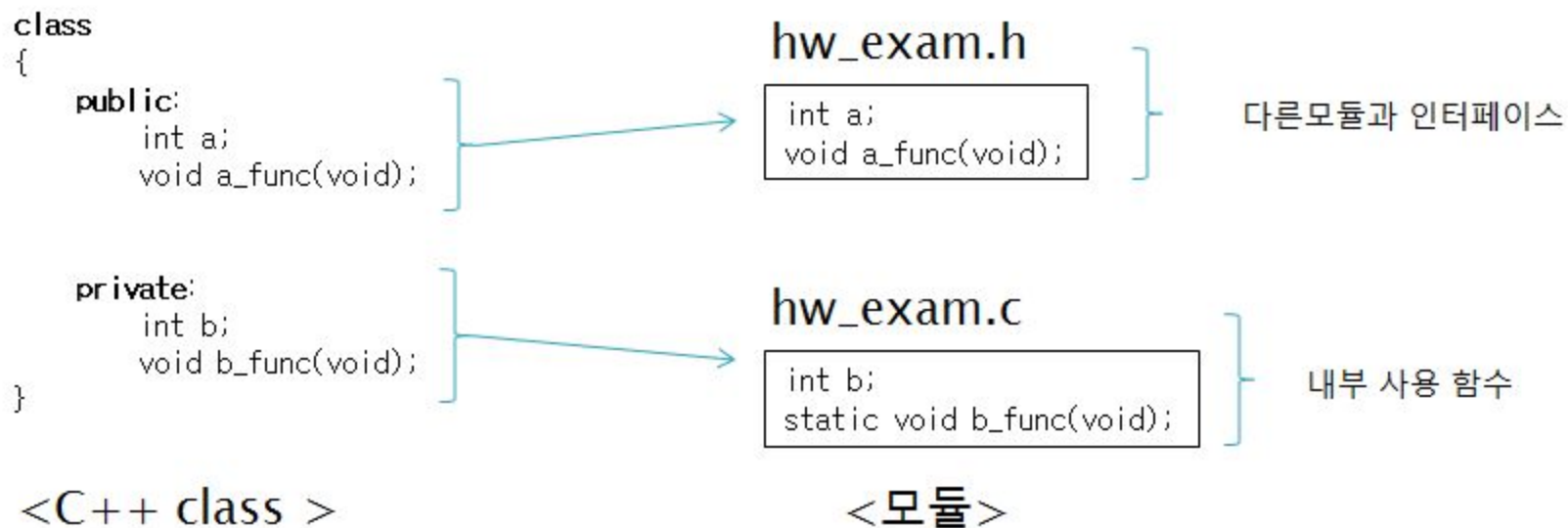
## uart.c

```
//-- Internal Functions
//
static bool uartIsEnable(uint8_t channel);
static void uartStartRx(uint8_t channel);
```

} 내부 사용 함수

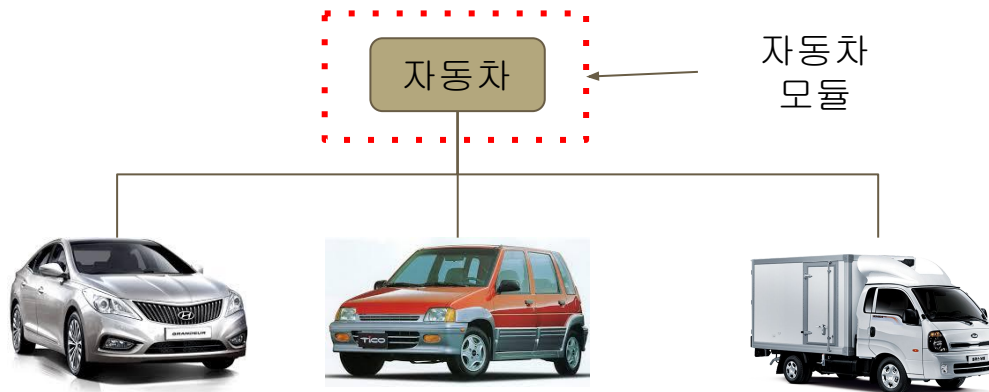
# 모듈화 설계

- C++ Class와 비교



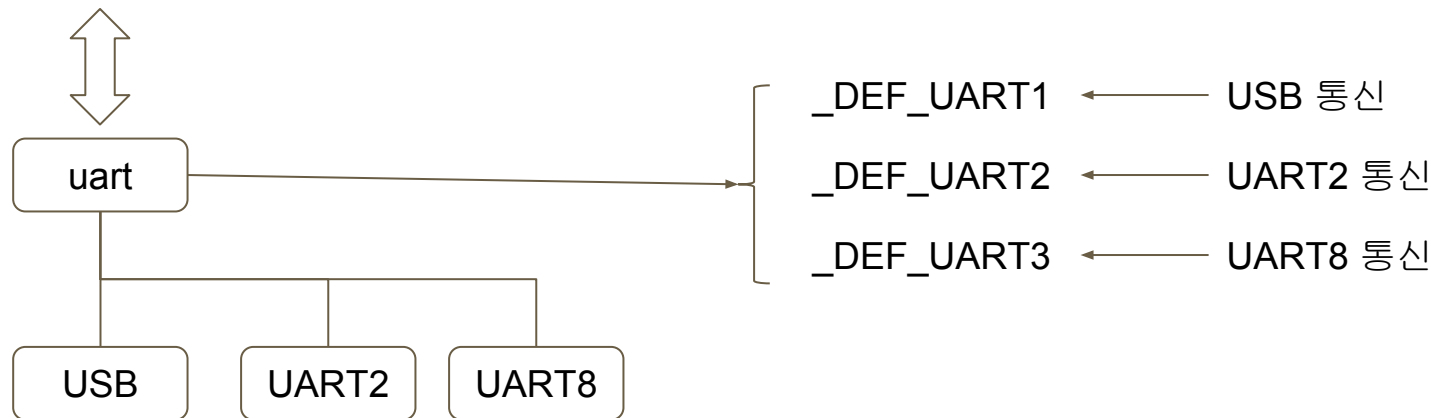
# 모듈화 설계

- 모듈을 만드는 기준
  - 모듈의 기능을 대표할 수 있을 정도로 추상화 시켜 모듈간 의존성 낮춤
  - 상위단에서는 최종으로 무엇이 사용되는지는 알 필요 없음



# 모듈화 설계

- 상위단 추상화에 의한 모듈간 의존성 제거
- 시리얼 통신 모듈 예
  - uart 모듈은 USB를 이용한 가상 시리얼과 하드웨어 UART로 구성되어 있고 사용하는 입장에서는 통신하는 채널이 USB인지 하드웨어 UART인지 구분할 필요 없음





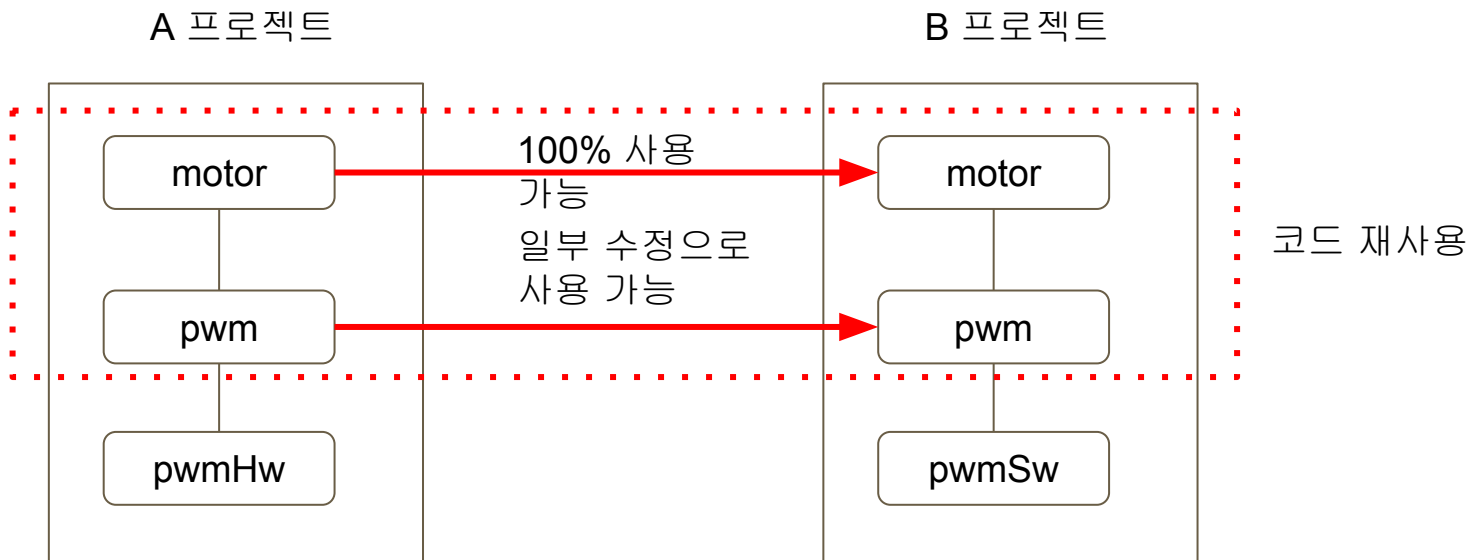
# 모듈화 설계

모듈을 분리하여 다른곳에  
사용하려고 할때?



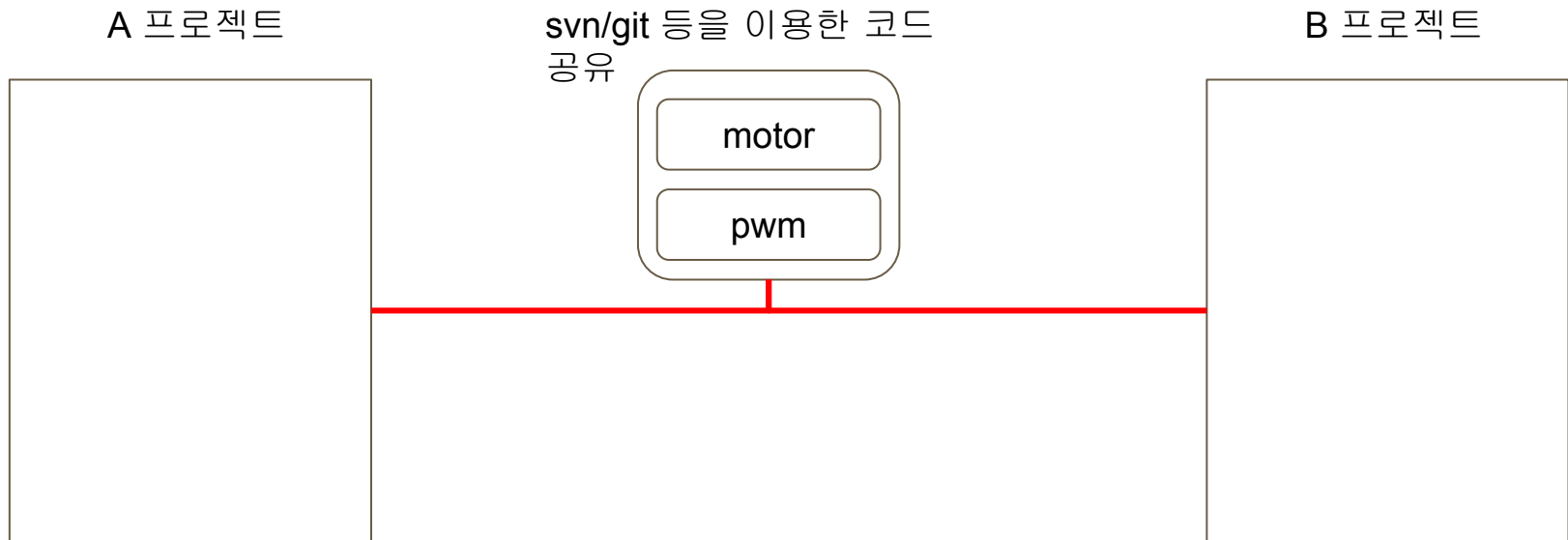
# 모듈화 설계

- 코드 재사용성 증가
- Layer가 증가할 수록 코드 재사용성은 증가
  - 반대로 Layer증가시 함수호출 오버헤드가 증가하고 복잡도가 증가 할 수 있음



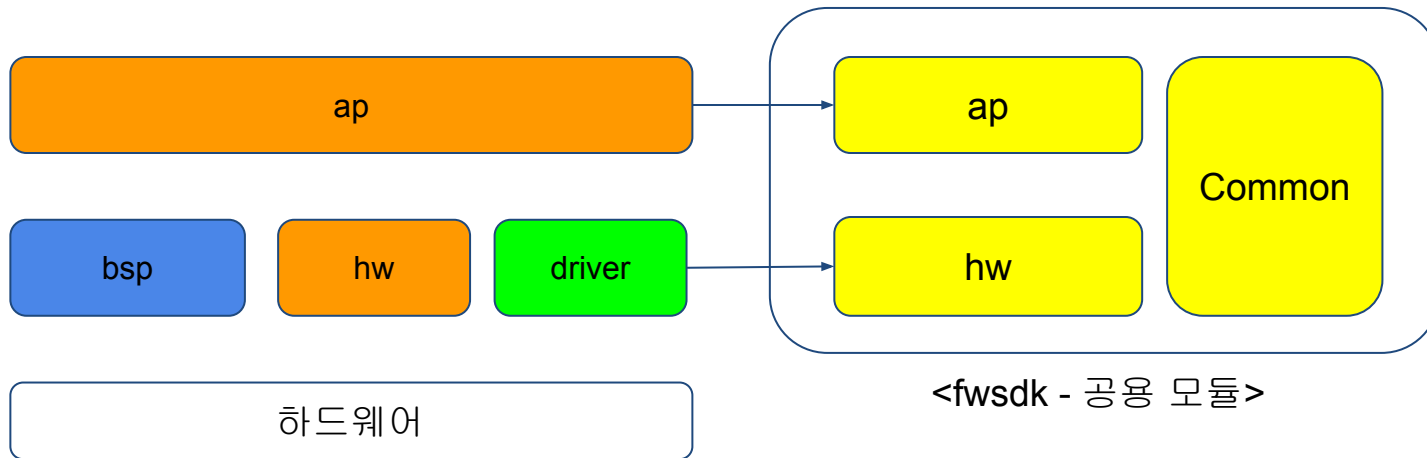
# 모듈화 설계

- 버전 관리 시스템을 이용하여 코드를 공유하여 중복 코드를 생성하지 않고 한번 개발하여 재사용성을 높임



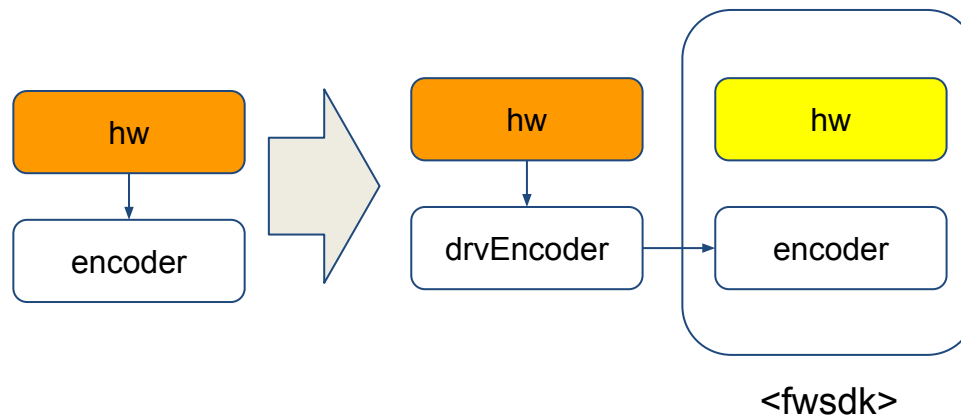
# 모듈화 설계 - OROCABOY

- driver Layer를 추가함
- fw sdk에 공용 API를 만들고, 각 프로젝트에 따라 driver를 구현함



# 모듈화 설계 - OROCABOY

- 초기 개발시에는 하드웨어 모듈 구현 후 공용 모듈로 driver로 분리 구현



- > ap
- > bsp
- ✓ fwsdk
  - > ap
  - > common
  - hw
- ✓ hw
  - core
  - driver
- > fatfs

# 모듈화 설계 - OROCABOY

fwsdk/hw/led.c

```
void ledToggle(uint8_t ch)
{
    bool led_state;

    led_state = drvLedGetState(ch);
    drvLedSetState(ch, !led_state);
}
```

project\_boot/hw/driverdrv\_led.c

```
bool drvLedGetState(uint8_t ch)
{
    GPIO_PinState pin_state = GPIO_PIN_RESET;
    bool ret = false;

    if (ch >= DRV_LED_MAX_CH) return false;

    pin_state = HAL_GPIO_ReadPin(drv_led_tbl[
    if (pin_state == GPIO_PIN_RESET) ret = tr

    return ret;
}
```

project\_app/hw/driverdrv\_led.c

```
bool drvLedGetState(uint8_t ch)
{
    GPIO_PinState pin_state = GPIO_PIN_RESET;
    bool ret = false;

    if (ch >= DRV_LED_MAX_CH) return false;

    pin_state = HAL_GPIO_ReadPin(drv_led_tbl[
    if (pin_state == GPIO_PIN_RESET) ret = tr

    return ret;
}
```

# 모듈화 설계 - OROCABOY

- 버전 관리 시스템을 이용하여 코드를 공유하여 중복 코드를 생성하지 않고 한번 개발하여 재사용성을 높임



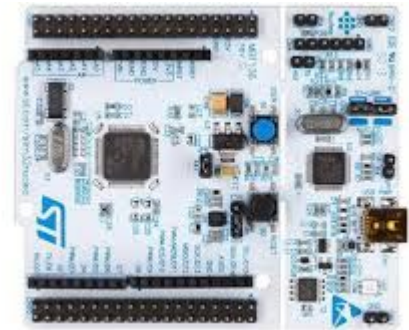
Time to break

새로운 MCU로 프로젝트를  
시작해야 한다면?



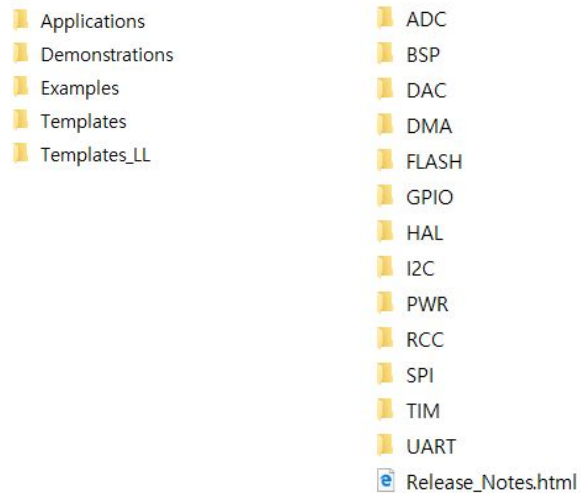
# Time to break

- 사용하고자 하는 MCU와 유사한 개발보드를 선정



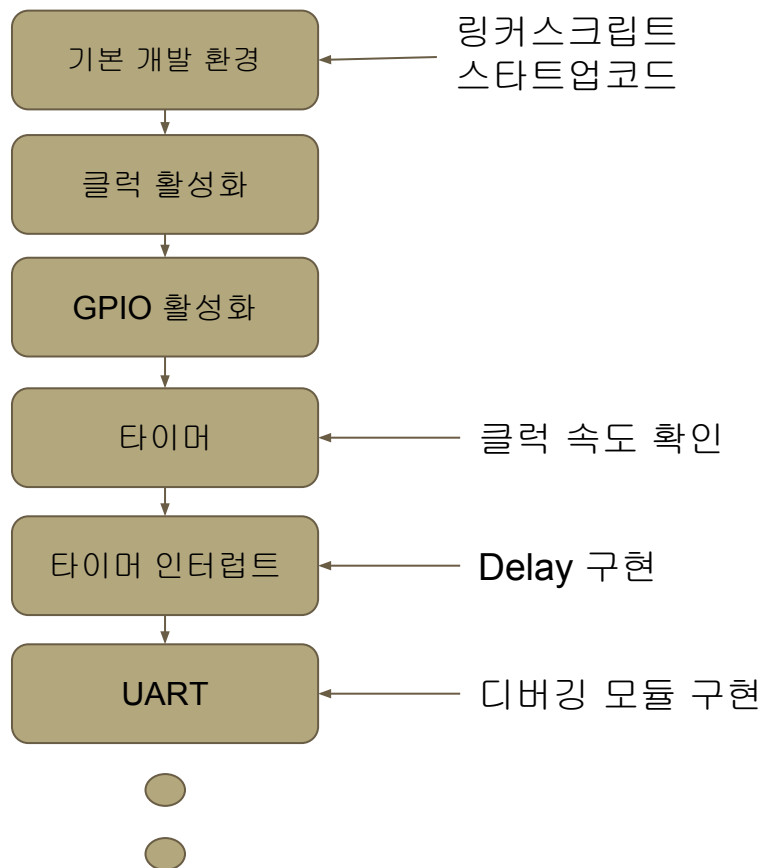
# Time to break

- 칩 제조사 제공 예제와 라이브러리 활용



# Time to break

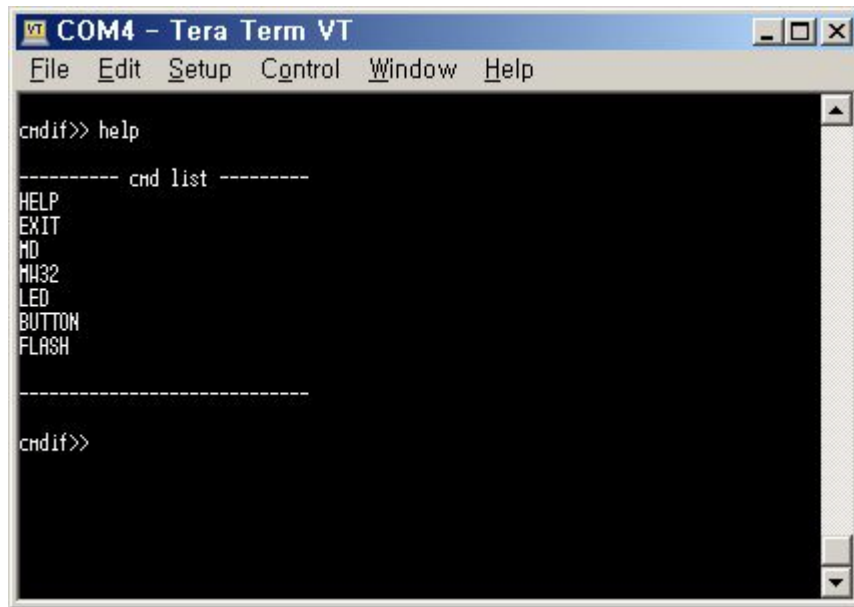
- 기본 개발 환경을 구축하고 순서대로 구현해가면서 시험



Ex01_Gpio
Ex01_Gpio_V2
Ex02_Uart
Ex03_Timer
Ex04_VCom
Ex05_I2C
Ex06_Radio
Ex07_GLcd
Ex08_GLcd_Dma
Ex09_I2C
Ex10_MPU6050
Ex11_Sonic
Ex12_HMC5883
...

# 디버깅 및 테스트

- Command 라인 방식의 디버깅 및 테스트 코드를 많이 사용함
- cmdif 모듈
  - 시리얼 통신 기반의 command 라인 방식 명령어 실행 모듈
  - 히스토리 기능으로 기존 실행한 이미지 사용 가능



```
COM4 - Tera Term VT
File Edit Setup Control Window Help

cmdif>> help

----- cmd list -----
HELP
EXIT
MD
MH32
LED
BUTTON
FLASH

-----

cmdif>>
```

# cmdif

- 초기화
  - cmdifInit()
    - 초기화 기능 수행
  - cmdifBegin()
    - 사용하고자 하는 시리얼 통신으로 초기화
  - cmdifLoop()
    - 초기화된 시리얼 통신으로 부터 데이터를 수신하여 명령어 실행

```
void hwInit(void)
{
    cmdifInit();
    swtimerInit();
    hwtimerInit();
    usbInit();
}
```

```
void mainInit(void)
{
    bspInit();
    hwInit();
    apInit();
    cmdifBegin(_DEF_UART1, 115200);
}
```

```
if (button_cnt == 3)
{
    cmdifPrint("cmdif begin \r\n");
    cmdifLoop();
    boot_excute = true;

    swtimerSet(led_timer_h, 100, LOOP_
}
```

# cmdif

- 기능 추가 방법

- cmdifAdd 함수를 사용하여 명령어를 추가함
- 추가된 명령어가 command 라인에 입력 되었을때 실행될 함수를 구현

```
void ledInit(void)
{
    GPIO_InitTypeDef GPIO_InitStructure;
    uint32_t i;

    for (i=0; i<LED_CH_MAX; i++)
    {
        GPIO_InitStructure.Pin = led_port_t
        GPIO_InitStructure.Mode = GPIO_MODE_
        HAL_GPIO_Init(led_port_tbl[i].por

        ledOff(i);
    }

    cmdifAdd("led", ledCmdif);
}
```



```
//-- ledCmdif
//
int ledCmdif(int argc, char **argv)
{
    bool ret = true;
    uint8_t number;

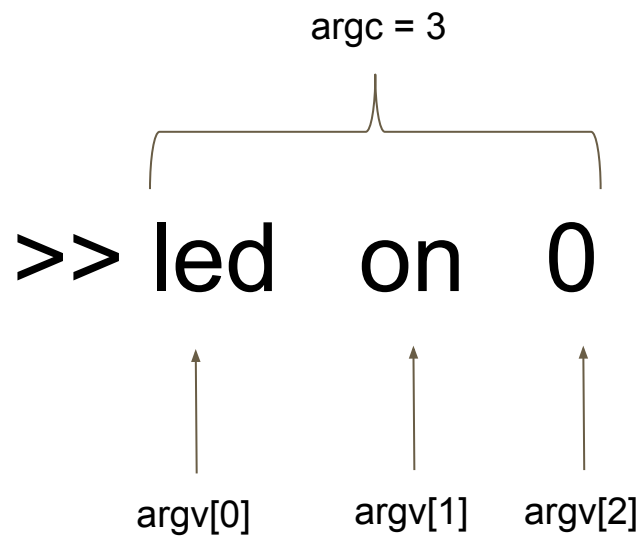
    if (argc == 3)
    {
        number = (uint8_t) strtoul((const char * )

        if(strcmp("on", argv[1]) == 0)
        {
            ledOn(number);
        }
        else if(strcmp("off", argv[1]) == 0)
        {
            ledOff(number);
        }
        else if(strcmp("toggle", argv[1]) == 0)
        {

```

# cmdif

- 명령어 파라미터
  - argc는 명령 파라미터 갯수
  - argv는 명령 파라미터 문자열 배열



# Mission

**Eclipse에서 빈 프로젝트를 생성 후  
예제 펌웨어를 복사하여 프로젝트  
옵션을 설정 후 디버깅까지 진행**