

# Mikrodenetleyici Driver Geliştirme

17 Mart 2023 Cuma 20:14

## Mikrodenetleyici Driver Geliştirme

### Kaynaklar

- Bu belge oluşturulurken <https://www.udemy.com/course/mikrodenetleyici-driver-gelistirme-gpio-spi-usart-i2c/> linkteki eğitim kursu izlenirken alınan notlardan oluşmaktadır.

### Giriş

- <https://usemynotes.com/what-are-header-files-in-c/>

Block	Name	Block base addresses	Size
Main memory	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
System memory		0x1FFF 0000 - 0x1FFF 77FF	30 Kbytes
OTP area		0x1FFF 7800 - 0x1FFF 7A0F	528 bytes
Option bytes		0x1FFF C000 - 0x1FFF C00F	16 bytes

- Main Memory, 0x0800 000'dan başlayarak 11 tane Sector'den oluşuyor ve toplam 1024KB yani 1MB alan flash için ayrılmış.

```
#define FLASH_BASE_ADR (0x08000000UL)
```

- Sondaki UL'nin anlamı unsigned long olduğunu belirtmek için kullanılmıştır.
- Bunu kullanmak yerine stdint.h kütüphanesini kullanarak başına uint32\_t yazabiliriz.

```
#define FLASH_BASE_ADR ((uint32_t)(0x08000000))
```

- SRAM adresi başlangıcı 0x20000000'dir. SRAM1, 112KB olduğundan SRAM2 için başlangıç adresini bulurken 112\*1024'ten 114688 buluruz ve hex formatına çevirirsek 1 C000 değerini SRAM adresine ekleriz.

```
#define SRAM1_BASE_ADDR (0x20000000UL)
```

```
#define SRAM2_BASE_ADDR (0x2001C000UL)
```

```
40 /*
5  * Memory Base Address
6  */
7
8 #define FLASH_BASE_ADDR (0x08000000UL) /* Flash Base Address (up to 1MB) */
9 #define SRAM1_BASE_ADDR (0x20000000UL) /* SRAM1 Base Address 112KB */
10 #define SRAM2_BASE_ADDR (0x2001C000UL) /* SRAM1 Base Address 16KB */
11
12
13 /*
14  * Peripheral Base Address
15  */
16
17 #define PERIP_BASE_ADDR (0x40000000UL) /* Base Address for All peripherals */
18
19 #define APB1_BASE_ADDR PERIP_BASE_ADDR /* APB1 Bus Domain Base Address */
20 #define APB2_BASE_ADDR (PERIP_BASE_ADDR + 0x100000UL) /* APB2 Bus Domain Base Address */
21 #define AHB1_BASE_ADDR (PERIP_BASE_ADDR + 0x200000UL) /* AHB1 Bus Domain Base Address */
22 #define AHB2_BASE_ADDR (PERIP_BASE_ADDR + 0x10000000UL) /* AHB2 Bus Domain Base Address */
```

```

25 // *
26 * APB1 Peripheral Base Address
27 */
28
29 #define TIM2_BASE_ADDR (APB1_BASE_ADDR + 0x0000UL)
30 #define TIM3_BASE_ADDR (APB1_BASE_ADDR + 0x0400UL)
31 #define TIM4_BASE_ADDR (APB1_BASE_ADDR + 0x0800UL)
32 #define TIM5_BASE_ADDR (APB1_BASE_ADDR + 0x0C00UL)
33 #define TIM6_BASE_ADDR (APB1_BASE_ADDR + 0x1000UL)
34 #define TIM7_BASE_ADDR (APB1_BASE_ADDR + 0x1400UL)
35 #define TIM12_BASE_ADDR (APB1_BASE_ADDR + 0x1800UL)
36 #define TIM13_BASE_ADDR (APB1_BASE_ADDR + 0x1C00UL)
37 #define TIM14_BASE_ADDR (APB1_BASE_ADDR + 0x2000UL)
38 #define RTC_BASE_ADDR (APB1_BASE_ADDR + 0x2800UL)
39 #define WWDG_BASE_ADDR (APB1_BASE_ADDR + 0x2C00UL)
40 #define IWDG_BASE_ADDR (APB1_BASE_ADDR + 0x3000UL)
41 #define I2S2ext_BASE_ADDR (APB1_BASE_ADDR + 0x3400UL)
42 #define SPI2_BASE_ADDR (APB1_BASE_ADDR + 0x3800UL)
43 #define SPI3_BASE_ADDR (APB1_BASE_ADDR + 0x3C00UL)
44 #define I2S3ext_BASE_ADDR (APB1_BASE_ADDR + 0x4000UL)
45 #define USART2_BASE_ADDR (APB1_BASE_ADDR + 0x4400UL)
46 #define USART3_BASE_ADDR (APB1_BASE_ADDR + 0x4800UL)
47 #define UART4_BASE_ADDR (APB1_BASE_ADDR + 0x4C00UL)
48 #define UART5_BASE_ADDR (APB1_BASE_ADDR + 0x5000UL)
49 #define I2C1_BASE_ADDR (APB1_BASE_ADDR + 0x5400UL)
50 #define I2C2_BASE_ADDR (APB1_BASE_ADDR + 0x5800UL)
51 #define I2C3_BASE_ADDR (APB1_BASE_ADDR + 0x5C00UL)
52 #define CAN1_BASE_ADDR (APB1_BASE_ADDR + 0x6400UL)
53 #define CAN2_BASE_ADDR (APB1_BASE_ADDR + 0x6800UL)
54 #define PWR_BASE_ADDR (APB1_BASE_ADDR + 0x7000UL)
55 #define DAC_BASE_ADDR (APB1_BASE_ADDR + 0x7400UL)

```

## GPIO

- Buraya kadar yapılan işlemler adresleme işlemiydi. Bu adresler tek başına tüm registra ulaşamazlar.
- Konfigürasyon işlemleri yapmak istiyorsak onun registarlarına **struct** yapısı altında tutarak ilerleyeceğiz.  
Bunu kullanırken **typedef** yapısını kullanarak yapacağız.
- Typedef ifadesini kullanarak, standart veri türlerinin (int, char, float, vs.) veya kullanıcı tanımlı yapıları farklı isimlerle tanımlayabiliriz. Bu şekilde mevcut bir veri türü için yeni bir isim veya yeni bir veri türü oluşturabiliriz.
- Veri türü ismindeki sondaki **\_t** bunun bir typedef olduğunu belirtmek için kullanırız.

```
#include <stdint.h>
```

```

typedef struct
{
    uint32_t MODER;
    uint32_t OTYPER;
    uint32_t OSPEEDER;
    uint32_t PUPDR;
    uint32_t ODR;
    uint32_t BSRR;
    uint32_t AFR[2];

```

```
}GPIO_TypeDef_t;
```

- GPIOA adresi üzerinden register'ın adresine ulaşmak için bir makro tanımladık.
- Bu makroda **\*** simgesi, bu ifadenin bir işaretçi olduğunu belirtir, yani **bellekte bir adresi** temsil ettiğini gösterir.
- Adres vasıtasıyla struct pointer kullanarak tek tek bu registarlarda ilerleyebiliriz yani her bir register için 4 byte şeklinde adresi ilerleyecek. Bu sebeple bu yapıyı kullanıyoruz.

```
#define GPIOA ((GPIO_TypeDef_t *) (GPIOA_BASE_ADDR))
```

- (GPIO\_TypeDef\_t \*) ifadesi verilen adresi GPIO\_TypeDef\_t türünde bir işaretçi olarak değerlendirir. Yani, GPIOA\_BASE\_ADDR adresindeki verilerin GPIO\_TypeDef yapısına uygun olduğu varsayılır.
- Main.c dosyamıza gelip registra yazma işlemi yapabiliriz.

```
#include "stm32f407xx.h"
```

- C'nin kendi kütüphanesi eklerken **< >** işareti kullanılmalı, kendi kütüphanemizi eklerken **" "** işareti kullanılmalıdır.
- Struct elemanlarına işaretçi üzerinden erişim yaptığımızdan -> ok işareti operatörünü kullandık. Doğrudan erişimlerde **.** nokta operatörünü kullanıyorduk.

```
int main(void)
```

```
{
```

```
GPIOA->
```

- AFR : uint32\_t [2]
- BSRR : uint32\_t
- MODER : uint32\_t
- ODR : uint32\_t
- OSPEEDER : uint32\_t
- OTYPER : uint32\_t
- PUPDR : uint32\_t

- <https://www.embedded.com/introduction-to-the-volatile-keyword/#:~:text=volatile%20is%20a%20qualifier%20that,code%20the%20compiler%20finds%20nearby>.
- [https://www.bilgigunlugum.net/prog/cprog/c\\_degisken](https://www.bilgigunlugum.net/prog/cprog/c_degisken) linkinde değişkenlerle beraber kullanılan extern, static, const, volatile gibi tanımlayıcılar hakkında bilgi edinebiliriz.
- **Extern**, bir değişkenin başka bir dosyada veya modülde tanımlandığını belirtir. İki dosya arasında değişkenin **paylaşılması** için kullanılır. Bir dosyada extern ile tanımlanan bir değişken, diğer dosyada bu değişkenin tanımlandığı yerin belirtilmesiyle kullanılabilir.
- **Static**, birkaç farklı amaç için kullanılır. Dosya içinde tanımlanan değişkenler, sadece tanımlandığı dosya içinde **geçerlidir** ve diğer dosyalar tarafından **erişilemez**. Bir fonksiyon içinde tanımlanan değişkenler, fonksiyon çağrıldığında ilk değerlerini alır ve fonksiyon **sona erse** bile değerlerini **korur**.
- **Const**, bir değişkenin değerinin programın çalışma süresi boyunca **değiştirilemeyeceğini** belirtir. Genellikle sabit değerlerin ve parametrelerin tanımlanmasında kullanılır. Const ifadesi yer kaplarken, Define metin tabanlı bir dönüştürme işlemi gerçekleştirdiğinden yer kaplamaz.
- **Volatile**, bir değişkenin değerinin herhangi bir zamanda dış etkenler tarafından **değiştirilebileceğini** belirtir.
- Volatile konusunda detaylı bilgi edinmek için <https://youtu.be/telqY-65kPk?si=TZ6ZM41IE7ooYXg3> linkteki videoyu izleyebiliriz.
- Bu tanımlayıcılar, değişkenlerin kapsamını, erişimini ve davranışını kontrol etmek için kullanılır. extern dosyalar arası bağlantıyı sağlar, static dosya içi kapsamı belirler, const sabit değerleri belirtir ve volatile değişkenin değerinin dış etkenler tarafından değiştirilebileceğini belirtir.

```
typedef struct
```

```
{
```

```
volatile uint32_t MODER;  
volatile uint32_t OTYPER;  
volatile uint32_t OSPEEDER;  
volatile uint32_t PUPDR;  
volatile uint32_t ODR;  
volatile uint32_t BSRR;  
volatile uint32_t AFR[2];
```

```
}GPIO_TypeDef_t;
```

```
#define __IO volatile
```

```
typedef struct
```

```
{
```

```
__IO uint32_t MODER;  
__IO uint32_t OTYPER;  
__IO uint32_t OSPEEDER;  
__IO uint32_t PUPDR;  
__IO uint32_t ODR;  
__IO uint32_t BSRR;  
__IO uint32_t AFR[2];
```

```
}GPIO_TypeDef_t;
```

- İyileştirme, derleyicinin, yazdığınız kodu daha hızlı çalıştırmak veya daha az bellek kullanmak gibi amaçlarla gerçekleştirdiği optimizasyonları ifade eder. Bu optimizasyonlar, derleyicinin ürettiği makine kodunun daha etkili ve performanslı olmasını sağlar.
- Bir değişkeni sık sık kullanılan bir hesaplama içinde geçici bir değer olarak kullanıyorsanız, derleyici bu değeri bir yazmaçta tutabilir. Bu, tekrar tekrar bellek üzerinde okuma ve yazma işlemi yapmak yerine daha hızlı erişim sağlar. Ancak, bu tür optimizasyonlar, belirli durumlarda yanıltıcı

olabilir, özellikle değişkenin değeri dış etkenler tarafından değiştirilebilecekse.

- Değişkenin başında volatile kullanılması durumunda derleyiciye değişkenin değerinin herhangi bir anda, derleyici tarafından gerçekleştirilmeyen bir işlemle değişebileceğini bildirir. Böylece derleyici, iyileştirme işlemi yapmaz ve değişkeni yazmaç yerine bellekten okunmasını sağlar. Kullanılmadığı durumda, derleyici iyileştirme işlemi uygular ve değişken değerinin değişmeyeceğini kabul ederek her defasında yazmaçtan okur.
- Bellek olarak ifade edilen RAM ya da donanım register'ı olabilir. Yazmaç ise genel amaçlı kayıtları (register) ifade eder.
- Bir değişken değerinin beklenmedik şekilde değişebildiği durumlarda, değişken volatile olarak bildirilmelidir. Böylece her zaman bellekteki en güncel değeri kullanması gerektiğini bildirir. Tüm çevresel birimlerin registerların değişkenleri volatile olarak tanımlanmalıdır. Kullanılmadığında kod hata vermez ama arka planda yanlış çalışacaktır. Çünkü derleyici bir kere değişkeni okuduktan sonra bir daha okumayacaktır ve değişken değiştiğinde bile eski değerini koruyacaktır.
- volatile kullanılmadığında, derleyici değişkeni yalnızca bir kez okuyup yazmaçta saklayabilir ve değişkenin değeri değişse bile eski değeri kullanmaya devam edebilir. volatile, her zaman güncel değerin kullanılmasını sağlar, böylece programın doğru çalışması garanti altına alınır.

```
86 /*
87  * General-purpose I/Os (GPIO)
88  */
89 typedef struct
90 {
91     __IO uint32_t MODER;          /*!< GPIO port mode register          Address offset: 0x00 */
92     __IO uint32_t OTYPER;        /*!< GPIO port output type register    Address offset: 0x04 */
93     __IO uint32_t OSPEEDER;      /*!< GPIO port output speed register   Address offset: 0x08 */
94     __IO uint32_t PUPDR;        /*!< GPIO port pull-up/pull-down register Address offset: 0x0C */
95     __IO uint32_t IDR;          /*!< GPIO port input data register     Address offset: 0x10 */
96     __IO uint32_t ODR;          /*!< GPIO port output data register    Address offset: 0x14 */
97     __IO uint32_t BSRR;         /*!< GPIO port bit set/reset register  Address offset: 0x18 */
98     __IO uint32_t LCKR;         /*!< GPIO port configuration lock register Address offset: 0x1C */
99     __IO uint32_t AFR[2];       /*!< GPIO alternate function low register Address offset: 0x20-0x24 */
100
101 }GPIO_TypeDef_t;
102
103 #define GPIOA ((GPIO_TypeDef_t *) (GPIOA_BASE_ADDR))
104 #define GPIOB ((GPIO_TypeDef_t *) (GPIOB_BASE_ADDR))
105 #define GPIOC ((GPIO_TypeDef_t *) (GPIOC_BASE_ADDR))
106 #define GPIOD ((GPIO_TypeDef_t *) (GPIOD_BASE_ADDR))
107 #define GPIOE ((GPIO_TypeDef_t *) (GPIOE_BASE_ADDR))
108
```

- Benzer yapıyı c dilinde yazarak uyguladık.

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>

typedef struct
{
    uint32_t MODER;
    uint32_t OTYPER;
    uint32_t OSPEEDER;
    uint32_t PUPDR;
    uint32_t IDR;
    uint32_t ODR;
    uint32_t BSRR;
    uint32_t LCKR;
    uint32_t AFR[2];
}GPIO_TypeDef_t;

int main(void)
{
    GPIO_TypeDef_t* GPIOA = (GPIO_TypeDef_t*)malloc(sizeof(GPIO_TypeDef_t));

    printf("Main      ADDR: %p\n\n", GPIOA);

    printf("MODER      ADDR: %p\n", &GPIOA->MODER);
    printf("OTYPER      ADDR: %p\n", &GPIOA->OTYPER);
    printf("OSPEEDER     ADDR: %p\n", &GPIOA->OSPEEDER);
    printf("PUPDR        ADDR: %p\n", &GPIOA->PUPDR);
    printf("IDR           ADDR: %p\n", &GPIOA->IDR);
    printf("ODR           ADDR: %p\n", &GPIOA->ODR);
    printf("BSRR         ADDR: %p\n", &GPIOA->BSRR);
}
```



```

printf("PUPDR ADDR: %p\n", &GPIOA->PUPDR);
printf("IDR ADDR: %p\n", &GPIOA->IDR);
printf("ODR ADDR: %p\n", &GPIOA->ODR);
printf("BSRR ADDR: %p\n", &GPIOA->BSRR);
printf("LCKR ADDR: %p\n", &GPIOA->LCKR);
printf("AFR[0] ADDR: %p\n", &GPIOA->AFR[0]);
printf("AFR[1] ADDR: %p\n", &GPIOA->AFR[1]);

return 0;
}

```

```

Main ADDR: 0000019D30CC9CB0
MODER ADDR: 0000019D30CC9CB0
OTYPER ADDR: 0000019D30CC9CB4
OSPEEDER ADDR: 0000019D30CC9CB8
PUPDR ADDR: 0000019D30CC9CBC
IDR ADDR: 0000019D30CC9CC0
ODR ADDR: 0000019D30CC9CC4
BSRR ADDR: 0000019D30CC9CC8
LCKR ADDR: 0000019D30CC9CCC
AFR[0] ADDR: 0000019D30CC9CD0
AFR[1] ADDR: 0000019D30CC9CD4

```

- Bit düzeyinde kullanılan dört ana mantıksal operatörü (AND, OR, XOR, NOT) ve bunların doğruluk tabloları aşağıdaki görselde yer almaktadır.
  - **Bitwise AND (&)** operatörü, bir registerdan bir biti okumak istiyorsak kullanıyoruz.
  - **Bitwise OR (|)** operatörü, registerdaki bir bite yazma işlemi yapmak istiyorsak kullanıyoruz.
  - **Bitwise XOR (^)** operatörü, registerdaki bir biti tersleme işlemi yapmak istiyorsak kullanıyoruz.
  - **Bitwise NOT (~)** operatörü, registerdaki bir biti temizleme işlemi yapmak istiyorsak kullanıyoruz.
- Bit kaydırma operatörleri, veriler üzerindeki bitleri belirli sayıda pozisyon sola veya sağa kaydırarak hızlı çarpma ve bölme işlemleri yapmayı sağlar.
  - **Sola Kaydırma (<<)**, bir sayının bitlerini sola doğru kaydırmak, sayıyı belirli bir sayıda ikiyle çarpmakla eşdeğerdir.
  - **Sağa Kaydırma (>>)** bir sayının bitlerini sağa doğru kaydırmak, sayıyı belirli bir sayıda ikiyle bölmekle eşdeğerdir.
- <https://youtu.be/Taik4BreiWw?si=ZAzh5ajnlbPLGDk> linkten bit işlemleri hakkında detaylı bilgi edinebiliriz.

```
#include <stdio.h>
#include <stdint.h>

int main(void)
{
    /*Logical Bitwise Operators*/
    uint8_t numberOne = 0x4; //0000 0100
    uint8_t numberTwo = 0x2; //0000 0010
    uint8_t result = 0;

    result = numberOne & numberTwo; //0000 0000
    printf("AND %#x=%d\n", result, result);

    result = numberOne | numberTwo; //0000 0110
    printf("OR  %#x=%d\n", result, result);

    result = numberOne ^ numberTwo; //0000 0110
    printf("XOR %#x=%d\n", result, result);

    result = ~numberOne; //1111 1011
    printf("NOT %#x=%d\n", result, result);

    printf("\n\n");

    /*Shift Operators*/
    uint32_t AHB1ENR = (0x1 << 5); //100000
    printf("AHB1ENR %#x=%d\n", AHB1ENR, AHB1ENR);

    AHB1ENR |= (0x1 << 4); //110000
    printf("AHB1ENR %#x=%d\n", AHB1ENR, AHB1ENR);

    AHB1ENR = (0x1 << 4); //010000
    printf("AHB1ENR %#x=%d\n", AHB1ENR, AHB1ENR);

    return 0;
}
```

```
AND 0=0
OR 0x6=6
XOR 0x6=6
NOT 0xfb=251
```

```
AHB1ENR 0x20=32
AHB1ENR 0x30=48
AHB1ENR 0x10=16
```

- Shift Operatör kullanımında yapılan sola kaydırmada 5.bite 1 yaz ve eşitle ardından tekrar yazma işlemi yapmak istersek önceki değerin kaybolmadan yeni yazma işlemi için OR operatörü kullanarak eşitleyerek yeni sayıya ulaşıyoruz. Eğer kullanmasaydık istenen bit dışındaki diğer bitleri sıfırlamış olacaktı.
- RCC için header ve surce dosyası oluşturuyoruz. Bu oluşturduğumuz dosyalara stm32f407xx.h dosyası dahil ediyoruz ve rcc.h dosyasını da main.c dosyasına dahil etmek yerine stm32f407xx.h dosyasına ekliyoruz.
- Registerdaki bitlere işlem yaparken kolaylık olması önceden makro olarak atıyoruz.
- Önce kaydetme işlemini yapan set, clear ve read adında tanımlama yapıyoruz.

```
#define SET_BIT(REG, BIT)      ((REG) |= (BIT))
#define CLEAR_BIT(REG, BIT)   ((REG) &= ~(BIT))
#define READ_BIT(REG, BIT)    ((REG) & (BIT))
```

- Daha sonra BIT kısmı için kaydırma işlemini yapan makroları yazıyoruz. Öncelikle GPIOA portunu aktif ediyoruz.

### RCC AHB1 peripheral clock register (RCC\_AHB1ENR)

[illegible]

- Bunun için AHB1 registerın 0.biti 1 yapmak için aşağıdaki tanımlamaları yapıyoruz.

```
#define RCC_AHB1ENR_GPIOAEN_Pos    (0U) // RCC AHB1ENR register GPIOAEN Bit Position
#define RCC_AHB1ENR_GPIOAEN_Msk    (0x1 << RCC_AHB1ENR_GPIOAEN_Pos) // RCC AHB1ENR register GPIOAEN Bit Mask
#define RCC_AHB1ENR_GPIOAEN        RCC_AHB1ENR_GPIOAEN_Msk // RCC AHB1ENR register GPIOAEN Macro
```

- Kullanıcının aktif etmesi için makro fonksiyonu yazıyoruz ve bunu do while ile yapıyoruz.
- Buradaki \ işareti aşağıdaki satırla devam ettiğini bildirir yoksa hata verir.
- tempValue değişkeni ile gecikme yapıyoruz ve bize hata vermemesi için UNUSED adında makro tanımlıyoruz.

```
#define UNUSED(x) (void)x
```

```
#define RCC_GPIOA_CLK_ENABLE()    do{ uint32_t tempValue = 0; \
                                   SET_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN); \
                                   tempValue = READ_BIT(RCC->AHB1ENR, RCC_AHB1ENR_GPIOAEN); \
                                   UNUSED(tempValue); \
                                   }while(0)
```

- Bunu main.c de yazıp çalıştırıyoruz ve Step Over tıkladıktan sonra ilgili bit 1 olmuş oluyor.

```
int main(void)
{
    RCC_GPIOA_CLK_ENABLE();
```

[illegible]

- Aktif edilen biti 0 yapmak için DISABLE makro fonksiyonu yazıyoruz.

```
#define RCC_GPIOA_CLK_DISABLE()    CLEAR_BIT(RCC->AHB1ENR,RCC_AHB1ENR_GPIOAEN);
```

```
int main(void)
{
    RCC_GPIOA_CLK_ENABLE();
    RCC_GPIOB_CLK_ENABLE();
    RCC_GPIOC_CLK_ENABLE();
    RCC_GPIOD_CLK_ENABLE();
    RCC_GPIOE_CLK_ENABLE();

    RCC_GPIOA_CLK_DISABLE();
}
```





```

#include <stdio.h>
#include <stdint.h>

#define DISABLE 0x0
#define ENABLE (!DISABLE)

#define GPIO_Pin_0 (0x0001)
#define GPIO_Pin_1 (0x0002)
#define GPIO_Pin_2 (0x0004)
#define GPIO_Pin_3 (0x0008)

void GPIO_WritePin(uint32_t* registerValue, uint8_t pinNumber, uint8_t
pinState)
{
    if (pinState == ENABLE)
    {
        *registerValue |= pinNumber;
    }
    else
    {
        *registerValue |= pinNumber << 16U;
    }
}

int main(void)
{
    uint32_t GPIO_BSRR_Register = 0x0;

    GPIO_WritePin(&GPIO_BSRR_Register, GPIO_Pin_1 | GPIO_Pin_2, ENABLE);

    printf("%d=0x%x\n", GPIO_BSRR_Register, GPIO_BSRR_Register);

    GPIO_WritePin(&GPIO_BSRR_Register, GPIO_Pin_1 | GPIO_Pin_2, DISABLE);

    printf("%d=0x%x\n", GPIO_BSRR_Register, GPIO_BSRR_Register);

    return 0;
}

```

6=0x6

393222=0x60006

- Gpio.h dosyasına aşağıdakileri tanımladık. gpio.c dosyasında tanımlanan fonksiyonun içeriğini yazıyoruz.

```

typedef enum
{
    GPIO_Pin_Reset = 0x0U,
    GPIO_Pin_Set = !GPIO_Pin_Reset
}GPIO_PinState t;

/*
 * @defgroup GPIO_pins_define
 */
#define GPIO_PIN_0 ((uint16_t)0x0001)
#define GPIO_PIN_1 ((uint16_t)0x0002)
#define GPIO_PIN_2 ((uint16_t)0x0004)
#define GPIO_PIN_3 ((uint16_t)0x0008)
#define GPIO_PIN_4 ((uint16_t)0x0010)
#define GPIO_PIN_5 ((uint16_t)0x0020)
#define GPIO_PIN_6 ((uint16_t)0x0040)
#define GPIO_PIN_7 ((uint16_t)0x0080)
#define GPIO_PIN_8 ((uint16_t)0x0100)
#define GPIO_PIN_9 ((uint16_t)0x0200)
#define GPIO_PIN_10 ((uint16_t)0x0400)
#define GPIO_PIN_11 ((uint16_t)0x0800)
#define GPIO_PIN_12 ((uint16_t)0x1000)
#define GPIO_PIN_13 ((uint16_t)0x2000)
#define GPIO_PIN_14 ((uint16_t)0x4000)
#define GPIO_PIN_15 ((uint16_t)0x8000)
#define GPIO_PIN_All ((uint16_t)0xFFFF)

```

```
void GPIO_WritePin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber, GPIO_PinState_t pinState);
```

```
/*
 * @brief GPIO_Write_Pin, makes pin High or Low
 * @note GPIOx = GPIO Port Base Address
 * @param pinNumber = GPIO Pin Numbers 0-15
 * @param pinState = GPIO_Pin_Set OR GPIO_Pin_Reset
 * @retval None
 */
void GPIO_WritePin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber, GPIO_PinState_t pinState)
{
    if(pinState == GPIO_Pin_Set)
    {
        GPIOx->BSRR = pinNumber;
    }
    else
    {
        GPIOx->BSRR = pinNumber << 16U;
    }
}
```

- Yazdığımız fonksiyona yorum satırı ekliyoruz. Bunlar **brief**, **param** ve **retval**'dir. Brief ile fonksiyonun ne yaptığını yazıyoruz. Param ile fonksiyonun parametrelerini yazıyoruz ve her parametre için ayrı ayrı param için de belirtiyoruz. Retval ile ne döndürdüğünü yazıyoruz.

```
GPIO_PinState_t GPIO_ReadPin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber);
```

```
/*
 * @brief GPIO_Read_Pin, makes the pin of GPIOx Port
 * @param GPIOx = GPIO Port Base Address
 * @param pinNumber = GPIO Pin Numbers 0-15
 * @retval GPIO_PinState_t
 */
GPIO_PinState_t GPIO_ReadPin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber)
{
    GPIO_PinState_t bitStatus = GPIO_Pin_Reset;

    if((GPIOx->IDR & pinNumber) != GPIO_Pin_Reset)
    {
        bitStatus = GPIO_Pin_Set;
    }
    return bitStatus;
}
```

```
void GPIO_LockPin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber);
```

```
/*
 * @brief GPIO_Lock_Pin, makes the pin of GPIOx Port
 * @param GPIOx = GPIO Port Base Address
 * @param pinNumber = GPIO Pin Numbers 0-15
 * @retval None
 */
void GPIO_LockPin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber)
{
    uint32_t tempValue = GPIO_LCKR_LCKK | pinNumber;

    GPIOx->LCKR = tempValue; /* Set LCKx bit LCKK='1' + LCK[15-0] */
    GPIOx->LCKR = pinNumber; /* Reset LCKx bit LCKK='0' + LCK[15-0] */
    GPIOx->LCKR = tempValue; /* Set LCKx bit LCKK='1' + LCK[15-0] */
    tempValue = GPIOx->LCKR; /* Read LCKR register */
}
```

- Pin ile ilgili özelliği kullanmak için registerda doğru biti aktif ederken pin sayısı kadar kaydırma yapmamız gerekiyor.
- Bunun için önceden yazdığımız pin tanımlamalarından yararlanacağız fakat pin tanımlamalarını direk kullanamıyoruz çünkü tanımlama yaparken değerlerini yazdırmıştık ve bu değer ile kaydırma işlemi yapamayız.
- Bu sebeple GPIO\_Init fonksiyonu ile pinin hangi pin olduğunu doğrulamamız gerekiyor. Bunu da 0.bite 1 yazıp sırasıyla diğer bitlere kaydırma yapıp sorgulama yapacak. Hangi bitte değer 1 ise pin o olmuş olacak.

```

#include <stdio.h>
#include <stdint.h>

#define GPIO_PIN_0      ((uint16_t)0x0001)
#define GPIO_PIN_1      ((uint16_t)0x0002)
#define GPIO_PIN_2      ((uint16_t)0x0004)
#define GPIO_PIN_3      ((uint16_t)0x0008)
#define GPIO_PIN_4      ((uint16_t)0x0010)
#define GPIO_PIN_5      ((uint16_t)0x0020)
#define GPIO_PIN_6      ((uint16_t)0x0040)
#define GPIO_PIN_7      ((uint16_t)0x0080)
#define GPIO_PIN_8      ((uint16_t)0x0100)
#define GPIO_PIN_9      ((uint16_t)0x0200)
#define GPIO_PIN_10     ((uint16_t)0x0400)
#define GPIO_PIN_11     ((uint16_t)0x0800)
#define GPIO_PIN_12     ((uint16_t)0x1000)
#define GPIO_PIN_13     ((uint16_t)0x2000)
#define GPIO_PIN_14     ((uint16_t)0x4000)
#define GPIO_PIN_15     ((uint16_t)0x8000)
#define GPIO_PIN_All    ((uint16_t)0xFFFF)

int main(void)
{
    uint16_t GPIO_Pin_Config = GPIO_PIN_11 | GPIO_PIN_3 | GPIO_PIN_12 | GPIO_PIN_1;

    for (uint16_t position = 0; position < 16U; position++)
    {
        uint16_t fakevalue = (0x1U << position);
        uint16_t isThere = (GPIO_Pin_Config) & fakevalue;

        if (fakevalue == isThere)
        {
            printf("Position:%d, fake value:0x%X, isThere:0x%X\n", position, fakevalue, isThere);
        }
    }

    return 0;
}

```

```

Position:1, fake value:0X2, isThere:0X2
Position:3, fake value:0X8, isThere:0X8
Position:11, fake value:0X800, isThere:0X800
Position:12, fake value:0X1000, isThere:0X1000

```

```

void GPIO_Init(GPIO_TypeDef_t *GPIOx, GPIO_InitTypeDef_t *GPIO_ConfigStruct);

```

```

3  /*
4  * @brief GPIO_Init, Configures the port and pin
5  * @param GPIOx = GPIO Port Base Address
6  * @param GPIO_InitTypeDef_t = User Config Structures
7  * @retval None
8  */
9  void GPIO_Init(GPIO_TypeDef_t *GPIOx, GPIO_InitTypeDef_t *GPIO_ConfigStruct)
10 {
11     uint32_t position;
12     uint32_t fakePosition = 0x00U;
13     uint32_t lastPosition = 0x00U;
14     uint32_t tempValue = 0x00U;
15
16     for(position=0; position < 16U; position++)
17     {
18         fakePosition = 0x1U << position;
19
20         lastPosition = (uint32_t)(GPIO_ConfigStruct->Pin) & fakePosition;
21
22         if(lastPosition == fakePosition)
23         {
24             /*----- GPIO Mode Configuration -----*/
25             /* Configure IO Direction mode (Input, Output, Alternate or Analog) */
26             tempValue = GPIOx->MODER;
27             tempValue &= ~(0x03U << (position * 2U));
28             tempValue |= (GPIO_ConfigStruct->Mode << (position * 2U));
29             GPIOx->MODER = tempValue;
30
31             if(GPIO_ConfigStruct->Mode == GPIO_MODE_OUTPUT || GPIO_ConfigStruct->Mode == GPIO_MODE_AF)
32             {
33                 /* Configure the IO Output Type */
34                 tempValue = GPIOx->OTYPER;
35                 tempValue &= ~(0x01U << position);
36                 tempValue |= (GPIO_ConfigStruct->Otype << position);
37                 GPIOx->OTYPER = tempValue;
38
39                 /* Configure the IO Speed */
40                 tempValue = GPIOx->OSPEEDER;
41                 tempValue &= ~(0x03U << (position * 2U));
42                 tempValue |= (GPIO_ConfigStruct->Speed << (position * 2U));
43                 GPIOx->OSPEEDER = tempValue;
44             }
45
46             /* Configure the IO Pull-up or Pull down*/
47             tempValue = GPIOx->PUPDR;
48             tempValue &= ~(0x03U << (position * 2U));
49             tempValue |= (GPIO_ConfigStruct->Pull << (position * 2U));
50             GPIOx->PUPDR = tempValue;
51         }
52     }
53 }

```

- Yazdığımız kodlar sonrası led blink uygulaması yapıyoruz.
- GPIO\_LedConfig() fonksiyonu başka c file'de kullanılmaması için static tanımladık.

```

1  #include "stm32f407xx.h"
2  #include "rcc.h"
3  #include "gpio.h"
4
5  static void GPIO_LedConfig();
6
7  int main(void)
8  {
9      GPIO_LedConfig();
10     GPIO_WritePin(GPIOD, GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15, GPIO_Pin_Set);
11
12     for(;;);
13 }
14
15 static void GPIO_LedConfig()
16 {
17     GPIO_InitTypeDef_t GPIO_LedStruct = { 0 };
18
19     RCC_GPIOD_CLK_ENABLE();
20
21     GPIO_LedStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
22     GPIO_LedStruct.Mode = GPIO_MODE_OUTPUT;
23     GPIO_LedStruct.Speed = GPIO_SPEED_LOW;
24     GPIO_LedStruct.Otype = GPIO_OTYPE_PP;
25     GPIO_LedStruct.Pull = GPIO_PUPD_NOPULL;
26
27     GPIO_Init(GPIOD, &GPIO_LedStruct);
28 }

```

```

void GPIO_TogglePin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber);

```



```

/*
 * @brief GPIO_Toggle_Pin, toggles the pin of GPIOx port
 * @param GPIOx = GPIO Port Base Address
 * @param pinNumber = GPIO Pin Numbers 0-15
 * @retval None
 */
void GPIO_TogglePin(GPIO_TypeDef_t *GPIOx, uint16_t pinNumber)
{
    uint32_t odr;

    odr = GPIOx->ODR;

    GPIOx->BSRR = ((odr & pinNumber) << 16U) | (~odr & pinNumber);
}

```

- GPIO\_InitTypeDef\_t ile tanımladığımız struct her biri uint32\_t tanımladığımız altı değişkenden toplam 24 byte yer kaplıyor.
- Her işlemde tekrar yeni bir struct oluşturmak yerine bir tane oluşturduğumuzu memset fonksiyonu ile temizleriz.

memset fonksiyonu kullanabilmek için string.h kütüphanesini kullanırız.

```

1 #include "stm32f407xx.h"
2 #include "rcc.h"
3 #include "gpio.h"
4
5 static void GPIO_Config();
6
7 int main(void)
8 {
9     GPIO_Config();
10
11     if(GPIO_ReadPin(GPIOA, GPIO_PIN_0) == GPIO_Pin_Set)
12     {
13         GPIO_WritePin(GPIOD, GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15, GPIO_Pin_Set);
14     }
15     else
16     {
17         GPIO_WritePin(GPIOD, GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15, GPIO_Pin_Reset);
18     }
19
20     for(;;);
21 }
22
23 static void GPIO_Config()
24 {
25     GPIO_InitTypeDef_t GPIO_InitStruct = { 0 };
26
27     RCC_GPIOD_CLK_ENABLE();
28     RCC_GPIOA_CLK_ENABLE();
29
30     GPIO_InitStruct.Pin = GPIO_PIN_12 | GPIO_PIN_13 | GPIO_PIN_14 | GPIO_PIN_15;
31     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT;
32     GPIO_InitStruct.Speed = GPIO_SPEED_LOW;
33     GPIO_InitStruct.Otype = GPIO_OTYPE_PP;
34     GPIO_InitStruct.Pull = GPIO_PUPD_NOPULL;
35     GPIO_Init(GPIOD, &GPIO_InitStruct);
36
37     memset(&GPIO_InitStruct, 0, sizeof(GPIO_InitStruct));
38
39     GPIO_InitStruct.Pin = GPIO_PIN_0;
40     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
41     GPIO_InitStruct.Pull = GPIO_PUPD_PULLDOWN;
42     GPIO_Init(GPIOA, &GPIO_InitStruct);
43 }

```

- GPIO pinleri, mikrodeneleyicilerde oldukça çok yönlüdür. Bu pinler sadece dijital giriş veya çıkış olarak değil, aynı zamanda UART, SPI, I2C gibi seri iletişim protokollerinde, PWM sinyallerinin üretilmesinde veya ADC gibi donanım işlevlerine erişimde kullanılabilir. Bir pinin bu tür özel işlevlere atanmasına **alternate function** denir.
- GPIO\_Init fonksiyonuna Alternate seçimi için gerekli ayarlamaları yapan kodu ekledik.

```

/* Configure the IO Alternate*/
if(GPIO_ConfigStruct->Mode == GPIO_MODE_AF)
{
    tempValue = GPIOx->AFR[position >> 3U];
    tempValue &= ~(0xFU << ((position & 0x7U) * 4U));
    tempValue |= (GPIO_ConfigStruct->Alternate << ((position & 0x7U) * 4U));
    GPIOx->AFR[position >> 3U] = tempValue;
}

```

- Her GPIO portu için birden fazla AFR vardır (örneğin AFR[0], AFR[1]), her biri 32 bit uzunluğunda

olup, her 4 bit bir pinin alternatif fonksiyonunu belirler.

- position >> 3U ile pin numarasını 8'e bölerek hangi AFR'nin kullanılacağını seçer. Pin numarası 12 ise 8'e bölümünden 1 çıkar böylece AFR[1] registerını kullanırız.
- Sonrasında pinin alternatif fonksiyon değerlerini temizleme işlemi yapılır. Uygun biti bulmak için position & 0x7U ile pin numarasının 8'e göre modu alınır ve 4 ile çarpılır. Pin numarası 12 ise 8'e modundan 4 çıkar ve bunu 4 ile çarparsak 16.bit olur.

```
/*
 * @defgroup GPIO_Pin_AF_define
 */
#define GPIO_AF0          (0x0U)
#define GPIO_AF1          (0x1U)
#define GPIO_AF2          (0x2U)
#define GPIO_AF3          (0x3U)
#define GPIO_AF4          (0x4U)
#define GPIO_AF5          (0x5U)
#define GPIO_AF6          (0x6U)
#define GPIO_AF7          (0x7U)
#define GPIO_AF8          (0x8U)
#define GPIO_AF9          (0x9U)
#define GPIO_AF10         (0xAU)
#define GPIO_AF11         (0xBU)
#define GPIO_AF12         (0xCU)
#define GPIO_AF13         (0xDU)
#define GPIO_AF14         (0xEU)
#define GPIO_AF15         (0xFU)
```

## External Interrupt

```
#define SYSCFG ((SYSCFG_TypeDef *) (SYSCFG_BASE_ADDR))
```

```
/*
 * External interrupt (EXTI)
 */
typedef struct
{
    __IO uint32_t IMR;           /*!< Interrupt mask register           Address offset: 0x00 */
    __IO uint32_t EMR;           /*!< Event mask register               Address offset: 0x04 */
    __IO uint32_t RTSR;          /*!< Rising trigger selection register  Address offset: 0x08 */
    __IO uint32_t FTSR;          /*!< Falling trigger selection register Address offset: 0x0C */
    __IO uint32_t SWIER;         /*!< Software interrupt event register Address offset: 0x10 */
    __IO uint32_t PR;            /*!< Pending register                  Address offset: 0x14 */
} EXTI_TypeDef_t;
```

```
#define EXTI ((EXTI_TypeDef *) (EXTI_BASE_ADDR))
```

- SYSCFG ayarı için için port ve pin bilgisi gerekiyor. Bunun için tanımlamaları yapıyoruz.

```
/*
 * @defgroup EXTI_GPIO_Port_define
 */
#define EXTI_GPIOA ((uint8_t)0x0)
#define EXTI_GPIOB ((uint8_t)0x1)
#define EXTI_GPIOC ((uint8_t)0x2)
#define EXTI_GPIOD ((uint8_t)0x3)
#define EXTI_GPIOE ((uint8_t)0x4)
#define EXTI_GPIOF ((uint8_t)0x5)
#define EXTI_GPIOG ((uint8_t)0x6)
#define EXTI_GPIOH ((uint8_t)0x7)
```

```
/*
 * @defgroup EXTI_Line_Pin_define
 */
#define EXTI_LINE_0 ((uint8_t)0x0)
#define EXTI_LINE_1 ((uint8_t)0x1)
#define EXTI_LINE_2 ((uint8_t)0x2)
#define EXTI_LINE_3 ((uint8_t)0x3)
#define EXTI_LINE_4 ((uint8_t)0x4)
#define EXTI_LINE_5 ((uint8_t)0x5)
#define EXTI_LINE_6 ((uint8_t)0x6)
#define EXTI_LINE_7 ((uint8_t)0x7)
#define EXTI_LINE_8 ((uint8_t)0x8)
#define EXTI_LINE_9 ((uint8_t)0x9)
#define EXTI_LINE_10 ((uint8_t)0xA)
#define EXTI_LINE_11 ((uint8_t)0xB)
#define EXTI_LINE_12 ((uint8_t)0xC)
#define EXTI_LINE_13 ((uint8_t)0xD)
#define EXTI_LINE_14 ((uint8_t)0xE)
#define EXTI_LINE_15 ((uint8_t)0xF)
```

- İşlemcide, interrupt yapılacak port ve pini belirtmek için bize 4 adet register tanımlanmış ve bu

her registerda 4 pin için yer ayrılmıştır. Bizim fonksiyon ile istenen pini doğru registera yazmamız gerekir. Bunun için pin numarasını 4'e bölerek doğru registerı bulabiliriz. Kalan ile ilgilenmiyoruz.

- Örnek verirse 7.pin için  $7/4=1$  sonucunu verir. Buradaki 1 kod kısmında CR[1]'e denk gelirken datasheette 2.registera denk gelir.
- Mod işlemi yapıp 4'e çarparsak ilgili pinin registerdaki bit numarasını bize verir.

```
int main(void)
{
    for (uint8_t pinNumber = 0; pinNumber < 16; pinNumber++) {
        printf("%d >> Register:%d, Bit:%d\n", pinNumber, pinNumber / 4, (pinNumber % 4) * 4);
    }

    return 0;
}
```

```
0 >> Register:0, Bit:0
1 >> Register:0, Bit:4
2 >> Register:0, Bit:8
3 >> Register:0, Bit:12
4 >> Register:1, Bit:0
5 >> Register:1, Bit:4
6 >> Register:1, Bit:8
7 >> Register:1, Bit:12
8 >> Register:2, Bit:0
9 >> Register:2, Bit:4
10 >> Register:2, Bit:8
11 >> Register:2, Bit:12
12 >> Register:3, Bit:0
13 >> Register:3, Bit:4
14 >> Register:3, Bit:8
15 >> Register:3, Bit:12
```

- Bu işlemi kaydırma yolu ile yapmak istersek sağa kaydırma ile bölme işlemi yapabiliriz. Eğer sayıyı sağa bir kez kaydırırsak 2'ye, iki kez kaydırırsak 4'e böler. Aslında her kaydırmada 2'nin üssü kadar bölme işlemi yapar. Eğer bunu sola kaydırırsak çarpma işlemi yapar.
- 2'nin katlarında mod işlemi için bölünecek sayının 2 ile üssü kadar 1 ile AND işlemi yapılır. Eğer 4'e böleceksem 2 üssü 2'den 11 yani 0x3 ile, 32'ye böleceksem 2 üssü 5'ten 11111 yani 0x1F uygulanır.

```
int main(void)
{
    for (uint8_t pinNumber = 0; pinNumber < 16; pinNumber++){
        uint8_t registerValue = pinNumber >> 2;
        uint8_t bitNumber = (pinNumber & 0x3) * 4;

        printf("%d >> Register:%d, Bit:%d\n", pinNumber, registerValue, bitNumber);
    }

    return 0;
}
```

```
/*
 * @brief GPIO_LineConfig, Configures the port and pin for SYSCFG
 * @param PortSource = Port Value A-H
 * @param LineSource = Pin Numbers
 * @retval None
 */
void EXTI_LineConfig(uint8_t PortSource, uint8_t EXTI_LineSource)
{
    uint32_t tempValue;

    tempValue = SYSCFG->EXTI_CR[EXTI_LineSource >> 2U];
    tempValue &= ~(0xFU << (EXTI_LineSource & 0x3U) * 4);
    tempValue = (PortSource << (EXTI_LineSource & 0x3U) * 4);
    SYSCFG->EXTI_CR[EXTI_LineSource >> 2U] = tempValue;
}
```

- Bu fonksiyonu çalıştırmadan önce SYSCFG'nin clock hattını aktif etmemiz gerekiyor.
- Öncesinde kütüphanelerde gerekli tanımlamaları yapıyoruz.

```
#define RCC_APB2ENR_SYSCFGEN_Pos (14U) // RCC APB2ENR register SYSCFGEN Bit Position
#define RCC_APB2ENR_SYSCFGEN_Msk (0x1 << RCC_APB2ENR_SYSCFGEN_Pos) // RCC APB2ENR register SYSCFGEN Bit Mask
#define RCC_APB2ENR_SYSCFGEN RCC_APB2ENR_SYSCFGEN_Msk // RCC APB2ENR register SYSCFGEN Macro
```



```
#define RCC_SYSCFG_CLK_ENABLE()    do{ uint32_t tempValue = 0; \
                                     SET_BIT(RCC->APB2ENR, RCC_APB2ENR_SYSCFGEN); \
                                     tempValue = SET_BIT(RCC->APB2ENR, RCC_APB2ENR_SYSCFGEN); \
                                     UNUSED(tempValue); \
                                     }while(0)
```

```
#define RCC_SYSCFG_CLK_DISABLE()      CLEAR_BIT(RCC->APB2ENR,RCC_APB2ENR_SYSCFGEN);
```

- C portun 7.pini için interrupt verdik.

```
RCC_SYSCFG_CLK_ENABLE();
```

```
EXTI_LineConfig(EXTI_GPIOC, EXTI_LINE_7);
```

- $7/4=1$ 'den EXT1 CR'nin 2.si, bit için kalan  $3*4$ 'den 12.bite denk geliyor C portu olduğundan 0010'dan ise 2 değerini yazdırıyor.
- Sonuç olarak ETICR2 registerın EXT17'nin 0010'dan 12.biti 2 olmalıdır.

SYSCFG			
MEMRM	0x40013800	0x0	
PMC	0x40013804	0x0	
EXTICR1	0x40013808	0x0	
EXTICR2	0x4001380c	0x2000	
EXTI7	[12:4]	0x2	
EXTI6	[8:4]	0x0	
EXTI5	[4:4]	0x0	
EXTI4	[0:4]	0x0	
EXTICR3	0x40013810	0x0	
EXTICR4	0x40013814	0x0	
CMPCR	0x40013820	0x0	

[illegible]

- SYSCFG yapısını tamamladık şimdi EXTI yapısının ayarlamasını yapacağız.

```
typedef struct
```

```
uint8_t EXTI_LineNumber; /*!> EXTI Line number for valid GPIO pin @ref EXTI_Line_Pin_define */
uint8_t TriggerSelection; /*!> EXTI TRigger @ref EXTI_Trigger */
uint8_t EXTI_Mode; /*!> EXTI Mode values @ref EXTI_Modes */
FunctionalState_t EXTI_LineCmd; /*!> Mask or Unmask the Line number */
```

```
}EXTI_InitTypeDef_t;
```

```
typedef enum
```

```
{
    DISABLE = 0x0U,
    ENABLE = !DISABLE
}
```

```
}FunctionalState t;
```

/\*

```
* @defgroup EXTI_Modes
```

```
#define EXTI_MODE_INTERRUPT (0x00U)
```

```
#define EXTI_MODE_EVENT (0x04U)
```

/ \*

```
* @defgroup EXTI_Trigger
```

```
#define EXTI_TRIGGER_NONE 0x00000000U
```

```
#define EXTI_TRIGGER_RISING 0x00000001U
```

```
#define EXTI_TRIGGER_FALLING 0x00000002U
```

```
#define EXTI_TRIGGER_RISING_FALLING (EXTI_TRIGGER_RISING | EXTI_TRIGGER_FALLING)
```



```

/*
 * @brief EXTI_Init for Valid GPIO port and Line Number
 * @param EXTI_InitStruct = User Config Structure
 * @retval None
 */
void EXTI_Init(EXTI_InitTypeDef_t *EXTI_InitStruct)
{
    uint32_t tempValue=0;
    tempValue = (uint32_t)EXTI_BASE_ADDR;

    EXTI->IMR &= ~(0x1U << EXTI_InitStruct->EXTI_LineNumber);
    EXTI->EMR &= ~(0x1U << EXTI_InitStruct->EXTI_LineNumber);

    if(EXTI_InitStruct->EXTI_LineCmd != DISABLE)
    {
        tempValue += EXTI_InitStruct->EXTI_Mode;

        *((__IO uint32_t*)tempValue) |= (0x1U << EXTI_InitStruct->EXTI_LineNumber);

        tempValue = (uint32_t)EXTI_BASE_ADDR;
        EXTI->RTSR &= ~(0x1U << EXTI_InitStruct->EXTI_LineNumber);
        EXTI->FTSR &= ~(0x1U << EXTI_InitStruct->EXTI_LineNumber);

        if(EXTI_InitStruct->TriggerSelection == EXTI_TRIGGER_RISING_FALLING)
        {
            EXTI->RTSR |= (0x1U << EXTI_InitStruct->EXTI_LineNumber);
            EXTI->FTSR |= (0x1U << EXTI_InitStruct->EXTI_LineNumber);
        }
        else
        {
            tempValue += EXTI_InitStruct->TriggerSelection;

            *((__IO uint32_t*)tempValue) |= (0x1U << EXTI_InitStruct->EXTI_LineNumber);
        }
    }
    else
    {
        tempValue = (uint32_t)EXTI_BASE_ADDR;

        tempValue += EXTI_InitStruct->EXTI_Mode;

        *((__IO uint32_t*)tempValue) &= ~(0x1U << EXTI_InitStruct->EXTI_LineNumber);
    }
}

```

```

#include <stdio.h>
#include <stdint.h>

int main(void)
{
    uint32_t variables[2] = { 1, 14 };
    uint32_t* pVariable = &variables[0];

    uint32_t temp = (uint32_t)pVariable;

    printf("Pointer:%p, TempValue:%X\n", pVariable, temp);

    temp += 0x4;

    *((uint32_t*)temp) = 22;

    printf("Pointer:%p, TempValue:%X\n", pVariable, temp);

    printf("Variable[1]=%d\n", variables[1]);

    return 0;
}

```

```

Pointer: 0000000061FE0C, TempValue:61FE0C
Pointer: 0000000061FE0C, TempValue:61FE10
Variable[1]=22

```

- Mikrodenetleyici tarafındaki ayarlamalarımızı tamamladık şimdi işlemci tarafının ayarlamalarını

yapacağız. Ayarlamaları nasıl yapacağımıza Arm Cortex-M4 Generic User Guide kitapçığının Peripheral bölümünün Interrupt kısmını inceliyoruz.

Address	Name	Type	Required privilege	Reset value	Description
0xE000E100-0xE000E11C	NVIC_ISER0-NVIC_ISER7	RW	Privileged	0x00000000	<i>Interrupt Set-enable Registers on page 4-4</i>
0xE000E180-0xE000E19C	NVIC_ICER0-NVIC_ICER7	RW	Privileged	0x00000000	<i>Interrupt Clear-enable Registers on page 4-5</i>
0xE000E200-0xE000E21C	NVIC_ISPR0-NVIC_ISPR7	RW	Privileged	0x00000000	<i>Interrupt Set-pending Registers on page 4-5</i>
0xE000E280-0xE000E29C	NVIC_ICPR0-NVIC_ICPR7	RW	Privileged	0x00000000	<i>Interrupt Clear-pending Registers on page 4-6</i>
0xE000E300-0xE000E31C	NVIC_IABR0-NVIC_IABR7	RW	Privileged	0x00000000	<i>Interrupt Active Bit Registers on page 4-7</i>
0xE000E400-0xE000E4EF	NVIC_IPR0-NVIC_IPR59	RW	Privileged	0x00000000	<i>Interrupt Priority Registers on page 4-7</i>
0xE000EF00	STIR	WO	Configurable <sup>a</sup>	0x00000000	<i>Software Trigger Interrupt Register on page 4-8</i>

- Kesmeyi aktif etmemiz gerekiyor. Bunun için NVIC\_ISER 0-7 registerlerini kullanacağız.

```
/*
 * Cortex-M4 Processor Peripherals
 */
#define NVIC_ISER0 ((uint32_t*)(0xE000E100UL))
```



Bits	Name	Function
[31:0]	SETENA	Interrupt set-enable bits. Write: 0 = no effect 1 = enable interrupt. Read: 0 = interrupt disabled 1 = interrupt enabled.

- Her bir bit bir kesmeyi tanımlıyor buna göre 256 kesme yapılacak alan bırakılmıştır fakat mikrodenetleyici de 82 adet kesme tanımlanmıştır. Hangi biti aktif edeceğimize mikrodenetleyici de tanımlanan kesmenin **Position** numarasından öğrenebiliriz.

Position	Priority	Type of priority	Acronym	Description	Address
6	13	settable	EXTI0	EXTI Line0 interrupt	0x0000 0058
7	14	settable	EXTI1	EXTI Line1 interrupt	0x0000 005C
8	15	settable	EXTI2	EXTI Line2 interrupt	0x0000 0060
9	16	settable	EXTI3	EXTI Line3 interrupt	0x0000 0064
10	17	settable	EXTI4	EXTI Line4 interrupt	0x0000 0068

```
/*
 * Interrupt Number Definition
 */
typedef enum
{
    EXTI0_IRQn = 6,          /*!< EXTI Line0 Interrupt */
    EXTI1_IRQn = 7,          /*!< EXTI Line1 Interrupt */
    EXTI2_IRQn = 8,          /*!< EXTI Line2 Interrupt */
    EXTI3_IRQn = 9,          /*!< EXTI Line3 Interrupt */
    EXTI4_IRQn = 10,         /*!< EXTI Line4 Interrupt */
}IRQn_TypeDef_t;
```

- Bu Position numarası **IRQ** numarası olarak geçiyor. Bunu 32'ye bölersem hangi registerda olduğunu bildiğimden çıkan sonucu başlangıç adresine eklerim.
- IRQ numaram 35 ise 32'ye bölümünde 1 çıkacaktır yani NVIC\_ISER1 registerına yazmam gerekiyor. Bu adrese gitmek için 1 eklediğimde başlangıç adresi olarak atadığım adresi 32 bit olarak

tanımladığımızdan 4 byte ekleme yapacaktır.

```
int main(int argc, char* argv[]) {  
  
    uint32_t myArr [] = {1, 3, 5, 12, 6, 58, 12};  
    uint32_t *pMyAddr = myArr;  
  
    for (int i = 0; i < 7; i++)  
    {  
        printf("%d -> %d\n", *(pMyAddr + i), (pMyAddr + i));  
    }  
  
    return 0;  
}
```

```
1 -> 6422000  
3 -> 6422004  
5 -> 6422008  
12 -> 6422012  
6 -> 6422016  
58 -> 6422020  
12 -> 6422024
```

```
/*  
 * @brief NVIC_EnableInterrupt  
 * @param IRQNumber = IRQ Number of Line  
 * @retval None  
 */  
void NVIC_EnableInterrupt(IRQn_TypeDef_t IRQNumber)  
{  
    uint32_t tempValue = 0;  
  
    tempValue = *((IRQNumber >> 5U) + NVIC_ISER0);  
    tempValue &= ~(0x1U << (IRQNumber & 0x1FU));  
    tempValue |= (0x1U << (IRQNumber & 0x1FU));  
    *((IRQNumber >> 5U) + NVIC_ISER0) = tempValue;  
}
```

- Artık main.c tarafında kod yazabiliriz. EXTI için gerekli ayarlamaları yapıyorum. Önceki örneklerde yaptığımız GPIO ayarlamalarını aynı şekilde kullanıyorum.

```
static void EXTI_Config()  
{  
    EXTI_InitTypeDef_t EXTI_InitStruct = { 0 };  
  
    RCC_SYSCFG_CLK_ENABLE();  
  
    EXTI_LineConfig(EXTI_GPIOA, EXTI_LINE_0);  
  
    EXTI_InitStruct.EXTI_LineCmd = ENABLE;  
    EXTI_InitStruct.EXTI_LineNumber = EXTI_LINE_0;  
    EXTI_InitStruct.EXTI_Mode = EXTI_MODE_INTERRUPT;  
    EXTI_InitStruct.TriggerSelection = EXTI_TRIGGER_RISING;  
  
    EXTI_Init(&EXTI_InitStruct);  
  
    NVIC_EnableInterrupt(EXTI0_IRQn);  
}
```

- Kesmeye girdiğinde hangi hat üzerinde geldiyse o fonksiyon çalışıyor. Bu fonksiyona startup dosyasından ulaşıyorum. Kesmeyi 0.pin olarak ayarladığımızdan EXTI0\_IRQHandler() fonksiyonunu kullanıyoruz.
- Kesmenin gelip gelmediğini Pending registarından anlıyorum. 0.bit, 1 ise kesme geldiğini anlamış oluyorum.
- Ardından sürekli kesmeye gitmemesi için clear işlemi yapmamız gerekiyor. Bu bite 1 yazıldığında programdan temizleniyor. Bu işlem donanımsal olduğundan bu şekilde çalışıyor. Bu sebeple 0.bite 1 yazıyorum.

```
void EXTI0_IRQHandler()
{
    if(EXTI->PR & 0x01)
    {
        EXTI->PR |= (0x01 << 0U);

        GPIO_WritePin(GPIOD, GPIO_PIN_12, GPIO_Pin_Set);
    }
}
```