

STM32 ile Gömülü Yazılım

5 Mayıs 2021 Çarşamba 07:50

- <https://www.udemy.com/course/stm32f4-discovery-kart-ile-arm-dersleri/>
- <https://github.com/mfatihkoseoglu/STM32F4-DISCOVERY>
- <https://www.udemy.com/course/stm32cubeide-ile-gomulu-yazlim-tasarim/>
- <https://www.kodlab.com/muhendislik/85-arm-mikrodenetleyiciler-9786052118269.html>

01 Clock

- 01_01 Sistem Saat Ayarlama

02 GPIO

- 02_01 Harici Led Yakma
- 02_02 Buton ile Led Yakma

03 External Interrupt

- 03_01 External Interrupt

04 ADC

- 04_01 ADC Verisi Okuma
- 04_02 ADC Interrupt

05 DAC

- 05_01 DAC Kullanımı
- 05_02 ADC Değeri İle DAC Kontrolü

06 DMA

- 06_01 DMA ile ADC Değer Okuma

07 Timer

- 07_01 Timer Değer Okuma
- 07_02 Timer Interrupt

08 PWM

- 08_01 PWM Kullanımı

09 USART

- 09_01 USART ile Mesaj Gönderme
- 09_02 USART ile Led Yakma

10 I2C

- 10_01 I2C Kullanımı

11 SPI

- 11_01 SPI Kullanımı

12 RNG

- 12_01 RNG Kullanımı

13 Flash Memory

- 13_01 Flash Memory Kullanımı

14 Örnek Projeler

- 14_01 ADXL345
- 14_02 MPU6050
- 14_03 BMP180

- 14_04 OLED
- 14_05 NRF24L01
- 14_06 SD KART
- 14_07 GPS
- 14_08 LORA
- 14_09 BME280
- 14_10 BMP280

01 Clock

5 Mayıs 2021 Çarşamba 08:02

01 Clock

01_01 Sistem Saat Ayarlama

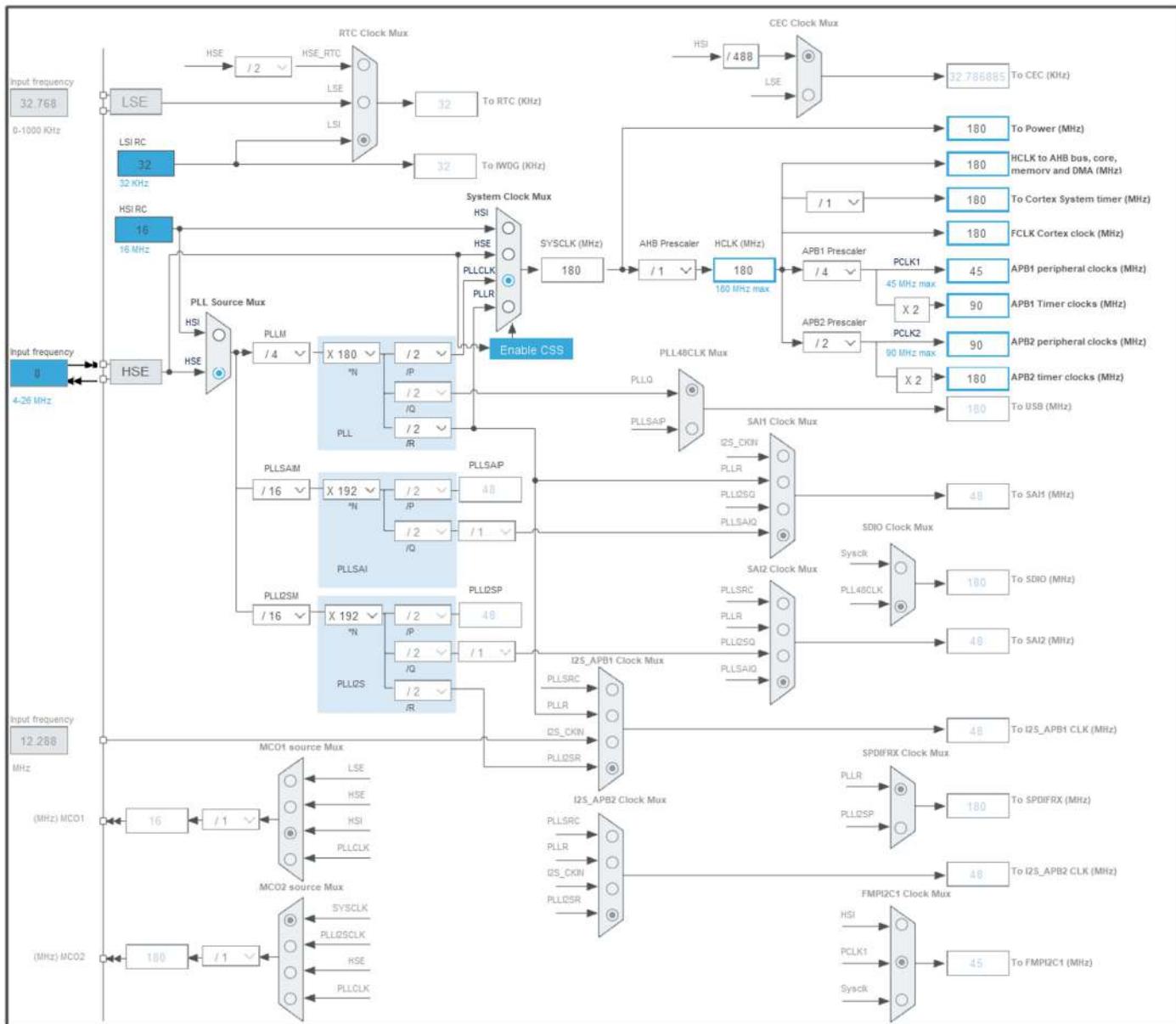
25 Aralık 2021 Cumartesi 00:51

01_01 Sistem Saat Ayarlama

➤ HAL

Konfigürasyon Kısımları

- Sistem saatini ayarlamak için öncelikle RCC kısmından HSE'den Crysal/Ceramic Resonator seçilir. Bu seçim işlemi ile beraber Clock Configuration da kutucuk bölme açılır.
LSE'nin HSE ile aralarındaki farkı düşük hızlı olması ve düşük güç tüketimi için kullanılır.
- System Clock Mux kısmında 3 seçenekimiz var. Bunlar HSI, HSE ve PLLCLK'tur. HSI dahili iken HSE harici kaynaktır.
HSI seçersek kutucuktaki değer olur, HSE seçersek giriş frekansı belli MHz aralıktadır. Biz burada kart üzerinde 8MHz olduğundan bu şekilde yapıp kullanıyoruz.
Eğer PLLCLK seçersek ayarlamalarla maksimum MHz'e kadar istediğimiz değeri seçebiliriz.



➤ SPL

Konfigürasyon Kısımlı Kod Kısımlı

➤ REGISTER

Konfigürasyon Kısımlı

- main.c dosyasındaki yorum satırları silip sadece hale getiriyoruz.

```

1 #include "stm32f4xx.h"
2
3 int main(void)
4 {
5     while (1)
6     {
7
8     }
9 }
```

- system_stm43f407.c dosyasındaki 182.satırda SystemCoreClock kısmına Ctrl ile sağ tıklıyoruz ve bizi system_stm43f407.h dosyasındaki 59.satırda götürüyor. Bu satırı kopyalayıp main.c dosyasına yapıştırıyoruz.

```

182     uint32_t SystemCoreClock = 168000000;

59 extern uint32_t SystemCoreClock;           /*!< System Clock Frequency (Core Clock) */

1 #include "stm32f4xx.h"
2
3 extern uint32_t SystemCoreClock;
4
5 uint32_t systemClock;
6
7@int main(void)
8 {
9     systemClock=SystemCoreClock;
10
11    while (1)
12    {
13    }
14 }
15 }

Expression          Type          Value
(x)= systemClock  uint32_t      168000000

```

```

1 #include "stm32f4xx.h"
2
3 extern uint32_t SystemCoreClock;
4
5 uint32_t systemClock;
6
7@int main(void)
8 {
9     systemClock=SystemCoreClock;          //168 000 000
10
11    RCC_DeInit();                      //HSI ON PLL OFF
12
13    SystemCoreClockUpdate();
14    systemClock=SystemCoreClock;          //16 000 000
15
16    while (1)
17    {
18    }
19 }
20 }

Expression          Type          Value
(x)= systemClock  uint32_t      160000000

```

- STM32F407VG mikrodenetleyicinin Reference Manuals'de RCC kısmına bakıyoruz. Mikrodenetleyici 32 bittir. Bu sebeple kaydediciler olan registerler 32 bittir.
- Adress offset bizim ilk olarak kaydedilen yeri gösterir. Reset value ise girilen değer sonrası tüm bitler sıfırlanır. Bit değerlerinde yazan r ile rw'nin anlamı r'nin read yani okunabilir, w'nin write yani yazılabılır anlamı vardır.
r olan yerlere müdahale edemiyoruz ama rw yazan bitlere müdahale edebiliyoruz.

RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

RCC clock control register (RCC_CR)

Address offset: 0x00

Reset value: 0x0000 XX83 where X is undefined.

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLI2S RDY	PLLI2S ON	PLL RDY	PLLON	Reserved				CSS ON	HSE BYP	HSE RDY	HSE ON
	r	rw	r	rw								rw	rw	r	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
HSICAL[7:0]								HSITRIM[4:0]				Res.	HSI RDY	HSION	
r	r	r	r	r	r	r	r	rw	rw	rw	rw		r	rw	

- "&=" anlam kaydet anlamındadır.
- Burada resetleme işlemi yapıyoruz.

`RCC->CR &= 0x00000083;`

- Harici osilatör kullanacağımdan HSEON olan 16.biti 1 yapmam gerekiyor.

Bit 16 **HSEON**: HSE clock enable

Set and cleared by software.

Cleared by hardware to stop the HSE oscillator when entering Stop or Standby mode. This bit cannot be reset if the HSE oscillator is used directly or indirectly as the system clock.

0: HSE oscillator OFF

1: HSE oscillator ON

- "|=" anlamı ise öncekine 1 ekle ve eşitle anlamındadır. Bunu kullanırken ekleme yani öteleme yapıyoruz.
- 16.biti 1 yapmak için öteleme kullanırken $1 \ll 16$ yazıyoruz yani 0.bitte 1 yazıp 16 bit öteliyor.

`RCC->CR |= 1 << 16;`

Bunu denetlemek için 17.biti kullanıyoruz. Yani 16.bit 1 olup olmadığını HSE osilatör 6 clock cycles yaptıktan sonra 17.bit olan HSERDY biti 1 oluyor.

Bit 17 **HSERDY**: HSE clock ready flag

Set by hardware to indicate that the HSE oscillator is stable. After the HSEON bit is cleared, HSERDY goes low after 6 HSE oscillator clock cycles.

0: HSE oscillator not ready

1: HSE oscillator ready

- While döngüsü 1 olduğu sürece çalışır. HSERDY biti 1 olana dek 0 olacağından ve döngünün çalışabilmesi için başına "!" işaretini koyarak tersliyoruz. 17.bit 1 olduğunda döngüden çıkışacaktır.
- & biti lojik kapılarda olduğu gibi 0&0 ile 0&1 olduğunda çıkışında 0 verir. 1&1 olduğunda çıkışında 1 verir.
- Buradaki $1 \ll 17$ işlemini yapabilmesi için 6 clock cycles zaman geçmesi gerekiyor. Öteleme işlem yaptığına anlamak için RCC->CR 1 olduğunda 17.bit 1 olmuş oluyor. Böylece 1&1'den 1 oluyor ve !1'den 0 olarak döngüden çıkışıyor.

`while(!(RCC->CR & (1 << 17)));`

- HSI kapatmak için HSION bitini 0 yapmam gerekiyor.

Bit 0 **HSION**: Internal high-speed clock enable

Set and cleared by software.

Set by hardware to force the HSI oscillator ON when leaving the Stop or Standby mode or in case of a failure of the HSE oscillator used directly or indirectly as the system clock. This bit cannot be cleared if the HSI is used directly or indirectly as the system clock.

0: HSI oscillator OFF
1: HSI oscillator ON

- $1 \ll 0$ ile 0.bit 1 yapılır ardından \sim işaretini ile 0 yapılır.

RCC->**CR** &= $\sim(1 \ll 0)$

Şu an mikrodenetleyici 8 000 000Hz'de çalışıyor. PLL ile 168 000 000Hz'e çıkaracağız.

Bit 19 **CSSON**: Clock security system enable

Set and cleared by software to enable the clock security system. When CSSON is set, the clock detector is enabled by hardware when the HSE oscillator is ready, and disabled by hardware if an oscillator failure is detected.

0: Clock security system OFF (Clock detector OFF)
1: Clock security system ON (Clock detector ON if HSE oscillator is stable, OFF if not)

- 19.bit 1 yapılır.

RCC->**CR** |= 1 << 19;

PLLON 24.bit 1 yapılmadan önce PLL'nin konfigürasyonlarını yapmamız gerekiyor.

RCC PLL configuration register (RCC_PLLCFGR)

Address offset: 0x04

Reset value: 0x2400 3010

Access: no wait state, word, half-word and byte access.

This register is used to configure the PLL clock outputs according to the formulas:

- $f_{(VCO\ clock)} = f_{(PLL\ clock\ input)} \times (PLL_N / PLL_M)$
- $f_{(PLL\ general\ clock\ output)} = f_{(VCO\ clock)} / PLL_P$
- $f_{(USB\ OTG\ FS,\ SDIO,\ RNG\ clock\ output)} = f_{(VCO\ clock)} / PLL_Q$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				PLLQ3	PLLQ2	PLLQ1	PLLQ0	Reserv ed	PLLSR C	Reserved				PLL_P1	PLL_P0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserv ed				PLLN						PLL_M5	PLL_M4	PLL_M3	PLL_M2	PLL_M1	PLL_M0
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- PLL için M, N ve P değerlerine göre sistem saatı 168MHz yapıyoruz. Bunun için M değerine 4, N değerine 168 ve P değerine 2 verdigimizde bu işlemi sağlamış oluyoruz.
- M için PLLM kısmında ilk 6 bitte yazacağımız. Bunun için 000100 değeri için 0., 1., 3., 4. ve 5.bite 0 yapıyorken 2.biti 1 yapıyoruz.
2.biti 1 yaparken ekleme yaparak yani " $|=$ " işaretini kullanırken diğerlerine sadece 0 bite kaydetmek için " $&=$ " işaretini kullanıyoruz.

Bits 5:0 **PLLM**: Division factor for the main PLL (PLL) and audio PLL (PLLI2S) input clock

Set and cleared by software to divide the PLL and PLLI2S input clock before the VCO.

These bits can be written only when the PLL and PLLI2S are disabled.

Caution: The software has to set these bits correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

VCO input frequency = PLL input clock frequency / PLLM with $2 \leq PLLM \leq 63$

00000000 - PLLM = 0 wrong configuration

Bits 5:0 **PLLM**: Division factor for the main PLL (PLL) and audio PLL (PLLI2S) input clock
Set and cleared by software to divide the PLL and PLLI2S input clock before the VCO.
These bits can be written only when the PLL and PLLI2S are disabled.

Caution: The software has to set these bits correctly to ensure that the VCO input frequency ranges from 1 to 2 MHz. It is recommended to select a frequency of 2 MHz to limit PLL jitter.

VCO input frequency = PLL input clock frequency / PLLM with $2 \leq PLLM \leq 63$

- 000000: PLLM = 0, wrong configuration
- 000001: PLLM = 1, wrong configuration
- 000010: PLLM = 2
- 000011: PLLM = 3
- 000100: PLLM = 4
- ...
- 111110: PLLM = 62
- 111111: PLLM = 63

```
RCC->PLLCFGR &= ~(1 << 0);           //PLLM0 0
RCC->PLLCFGR &= ~(1 << 1);           //PLLM1 0
RCC->PLLCFGR |= (1 << 2);            //PLLM2 1
RCC->PLLCFGR &= ~(1 << 3);           //PLLM3 0
RCC->PLLCFGR &= ~(1 << 4);           //PLLM4 0
RCC->PLLCFGR &= ~(1 << 5);           //PLLM5 0
```

N için PLLN kısmını kullanıyoruz.

Bits 14:6 **PLLN**: Main PLL (PLL) multiplication factor for VCO
Set and cleared by software to control the multiplication factor of the VCO. These bits can be written only when PLL is disabled. Only half-word and word accesses are allowed to write these bits.

Caution: The software has to set these bits correctly to ensure that the VCO output frequency is between 100 and 432 MHz.

VCO output frequency = VCO input frequency \times PLLN with $50 \leq PLLN \leq 432$

- 000000000: PLLN = 0, wrong configuration
- 000000001: PLLN = 1, wrong configuration
- ...
- 000110010: PLLN = 50
- ...
- 001100011: PLLN = 99
- 001100100: PLLN = 100
- ...
- 110110000: PLLN = 432
- 110110001: PLLN = 433, wrong configuration
- ...
- 111111111: PLLN = 511, wrong configuration

Note: Multiplication factors ranging from 50 and 99 are possible for VCO input frequency higher than 1 MHz. However care must be taken that the minimum VCO output frequency respects the value specified above.

HEX	A8
DEC	168
OCT	250
BIN	1010 1000

168 decimal sayısının binary karşılığı 1010 1000'dir. Biz burada 9 bit olduğundan 010101000 yazacağız.

Biz bununla uğraşmak yerine 6 bit öteleyip 168 sayısını yazacağız.

```
RCC->PLLCFGR |= (168 << 6);           //PLLN 168
```

Aynı işlemi M için de yapabiliriz.

```
RCC->PLLCFGR |= (4 << 0);           //PLLM 4
```

P için PLLP kısmını kullanıyoruz. 2 değeri için iki biti 0 yap diyor.

Bits 17:16 **PLLP**: Main PLL (PLL) division factor for main system clock

Set and cleared by software to control the frequency of the general PLL output clock. These bits can be written only if PLL is disabled.

Caution: The software has to set these bits correctly not to exceed 168 MHz on this domain.

PLL output clock frequency = VCO frequency / PLLP with PLLP = 2, 4, 6, or 8

00: PLLP = 2

01: PLLP = 4

10: PLLP = 6

11: PLLP = 8

```
RCC->PLLCFGR &= ~(1 << 16);           //PLLP1 0  
RCC->PLLCFGR &= ~(1 << 17);           //PLLP2 0
```

- PLL için başlangıçta tüm bitleri 0 yapıyoruz.
 - Burda 8 tane sıfır var. Buradaki her biri 32 bit'de 4 bite karşılık geliyor.
- RCC->CFGGR = 0x00000000;
- Tüm bitleri başlangıçta 0 yaptığımız için PLL'de P için yaptığımız 0'lama işlemine gerek kalmadı.

```
7 void RCC_Config(void)  
8 {  
9     //RCC->CR &= 0x00000083;           //RESET  
10  
11    RCC->CR &= ~(1 << 0);           //HSION  
12    RCC->CR |= 1 << 16;              //HSEON  
13    while(!(RCC->CR & (1 << 17))); //HSERDY  
14    RCC->CR |= 1 << 19;              //CSSON  
15    RCC->CFGGR = 0x00000000;  
16    //RCC->PLLCFGR &= ~(1 << 0);    //PLLM0 0  
17    //RCC->PLLCFGR &= ~(1 << 1);    //PLLM1 0  
18    //RCC->PLLCFGR |= (1 << 2);      //PLLM2 1  
19    //RCC->PLLCFGR &= ~(1 << 3);    //PLLM3 0  
20    //RCC->PLLCFGR &= ~(1 << 4);    //PLLM4 0  
21    //RCC->PLLCFGR &= ~(1 << 5);    //PLLM5 0  
22    RCC->PLLCFGR |= (4 << 0);       //PLLM 4  
23    RCC->PLLCFGR |= (168 << 6);     //PLLN 168  
24    //RCC->PLLCFGR &= ~(1 << 16);    //PLLP1 0  
25    //RCC->PLLCFGR &= ~(1 << 17);    //PLLP2 0  
26 }
```

Registers			
type filter text			
Register	Address	Value	
> DMA1			
`> RCC			
> CR	0x40023800	0xb7083	
`> PLLCFGR	0x40023804	0xa04	
`> PLLQ3	[27:1]	0x0	
`> PLLQ2	[26:1]	0x0	
`> PLLQ1	[25:1]	0x0	
`> PLLQ0	[24:1]	0x0	
`> PLLSRC	[22:1]	0x0	
`> PLLP1	[17:1]	0x0	
`> PLLP0	[16:1]	0x0	
`> PLLN8	[14:1]	0x0	
`> PLLN7	[13:1]	0x1	
`> PLLN6	[12:1]	0x0	
`> PLLN5	[11:1]	0x1	
`> PLLN4	[10:1]	0x0	
`> PLLN3	[9:1]	0x1	
`> PLLN2	[8:1]	0x0	
`> PLLN1	[7:1]	0x0	
`> PLLN0	[6:1]	0x0	
`> PLLM5	[5:1]	0x0	
`> PLLM4	[4:1]	0x0	
`> PLLM3	[3:1]	0x0	
`> PLLM2	[2:1]	0x1	
`> PLLM1	[1:1]	0x0	
`> PLLM0	[0:1]	0x0	

- PLLSRC ile PLL sürücüsü seçilir. Bu bit için HSI kullanırsak 0, HSE kullanırsak 1 yapılır. Biz HSE kullandığımızdan bu biti 1 yapıyoruz.

Bit 22 **PLLSRC**: Main PLL(PLL) and audio PLL (PLLI2S) entry clock source

Set and cleared by software to select PLL and PLLI2S clock source. This bit can be written only when PLL and PLLI2S are disabled.

0: HSI clock selected as PLL and PLLI2S clock entry

1: HSE oscillator clock selected as PLL and PLLI2S clock entry

```
RCC->CFGREG |= (1 << 22);
```

Bit 24 **PLLON**: Main PLL (PLL) enable

Set and cleared by software to enable PLL.

Cleared by hardware when entering Stop or Standby mode. This bit cannot be reset if PLL clock is used as the system clock.

0: PLL OFF

1: PLL ON

```
RCC->CR |= 1 << 24;
```

Bit 25 **PLLRDY**: Main PLL (PLL) clock ready flag

Set by hardware to indicate that PLL is locked.

0: PLL unlocked

1: PLL locked

```
while(!(RCC->CR & (1 << 25)));
```

RCC clock configuration register (RCC_CFGREG)

Address offset: 0x08

Reset value: 0x0000 0000

Access: 0 ≤ wait state ≤ 2, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]			MCO1 PRE[2:0]			I ₂ SSC R	MCO1		RTCPRE[4:0]				
rw		rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved	HPRE[3:0]				SWS1	SWS0	SW1	SW0	
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	r	r	rw	rw	

- İlk önce 0.bit 0 yapıldı daha sonra 1.bit 1 yapılarak sistemin saatini PLL ile ayarladığımızı belirtiyoruz.

Bits 1:0 **SW**: System clock switch

Set and cleared by software to select the system clock source.

Set by hardware to force the HSI selection when leaving the Stop or Standby mode or in case of failure of the HSE oscillator used directly or indirectly as the system clock.

00: HSI oscillator selected as system clock

01: HSE oscillator selected as system clock

10: PLL selected as system clock

11: not allowed

```
RCC->CFGREG &= ~(1 << 0);
```

```
RCC->CFGREG |= (1 << 1);
```

Bits 3:2 **SWS**: System clock switch status

Set and cleared by hardware to indicate which clock source is used as the system clock.

00: HSI oscillator used as the system clock

01: HSE oscillator used as the system clock

10: PLL used as the system clock

11: not applicable

```

while(!(RCC->CR & (1 << 1)));
Kod Kismi
7 void RCC_Config(void)
8 {
9     //RCC->CR &= 0x00000083;           //RESET
10
11    RCC->CR &= ~(1 << 0);          //HSION
12    RCC->CR |= 1 << 16;            //HSEON
13    while(!(RCC->CR & (1 << 17))); //HSERDY
14    RCC->CR |= 1 << 19;            //CSSON
15    RCC->CFGR = 0x00000000;
16    RCC->PLLCFGR |= (1 << 22);    //PLLSRC
17    //RCC->PLLCFGR &= ~(1 << 0);   //PLLM0 0
18    //RCC->PLLCFGR &= ~(1 << 1);   //PLLM1 0
19    //RCC->PLLCFGR |= (1 << 2);    //PLLM2 1
20    //RCC->PLLCFGR &= ~(1 << 3);   //PLLM3 0
21    //RCC->PLLCFGR &= ~(1 << 4);   //PLLM4 0
22    //RCC->PLLCFGR &= ~(1 << 5);   //PLLM5 0
23    RCC->PLLCFGR |= (4 << 0);      //PLLM 4
24    RCC->PLLCFGR |= (168 << 6);    //PLLN 168
25    //RCC->PLLCFGR &= ~(1 << 16);  //PLLP1 0
26    //RCC->PLLCFGR &= ~(1 << 17);  //PLLP2 0
27
28    RCC->CR |= 1 << 24;            //PLLON
29    while(!(RCC->CR & (1 << 25))); //PLLRDY
30
31    RCC->CFGR &= ~(1 << 0);
32    RCC->CFGR |= (1 << 1);         //SW
33
34    while(!(RCC->CR & (1 << 1))); //SWS
35 }
36 int main(void)
37 {
38     //systemClock=SystemCoreClock;    //168 000 000
39
40     //RCC_DeInit();                //HSI ON PLL OFF
41
42     //SystemCoreClockUpdate();
43     //systemClock=SystemCoreClock;    //16 000 000
44
45     RCC_Config();
46     SystemCoreClockUpdate();
47     systemClock=SystemCoreClock;    //168 000 000
48
49     while (1)
50     {
51
52     }
53 }
54 }
```

Expression	Type	Value
(*) systemClock	uint32_t	168000000

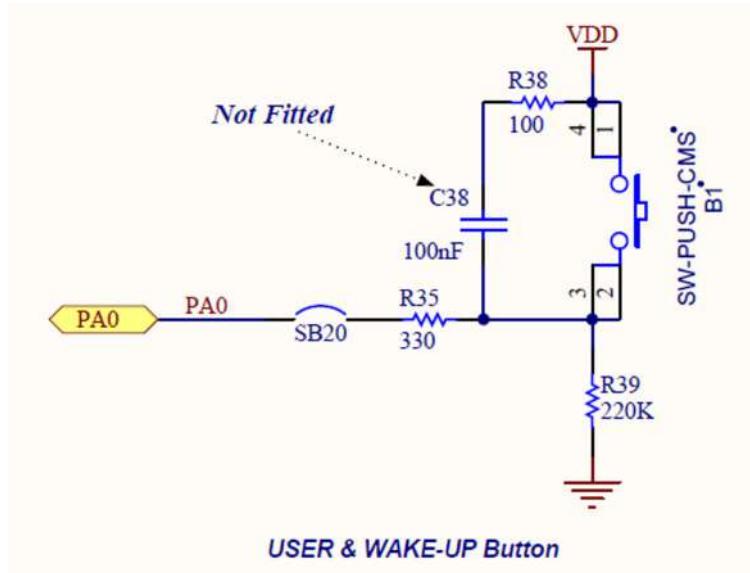
02 GPIO

5 Mayıs 2021 Çarşamba 08:02

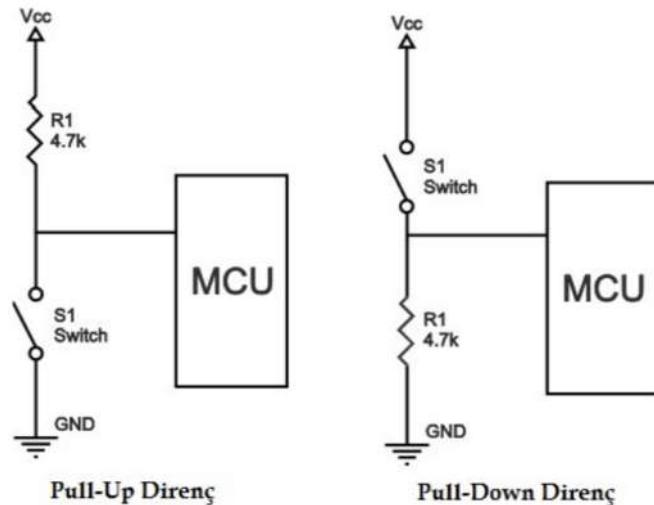
02 GPIO

Giriş

- Butonlar ve anahtarlar mikrodenetleyiciye giriş pini üzerinden lojik 1 ve lojik 0 olarak bilgi girişini sağlayan mekanik elemanlardır.

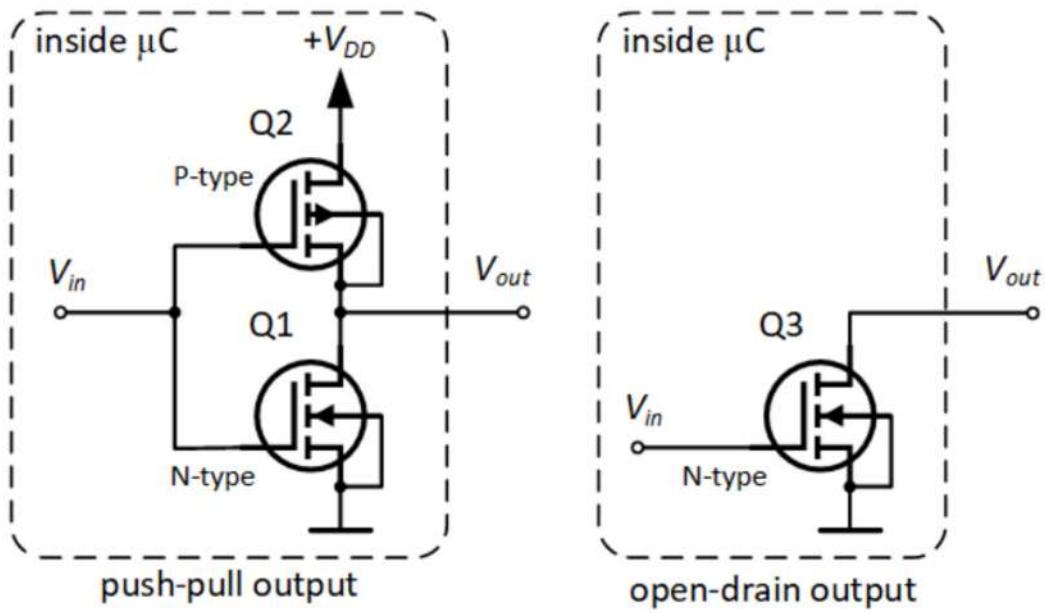


- Resimde görüldüğü gibi kullanıcı butonu A portunun 0. pinine bağlı ve pull down durumundadır.

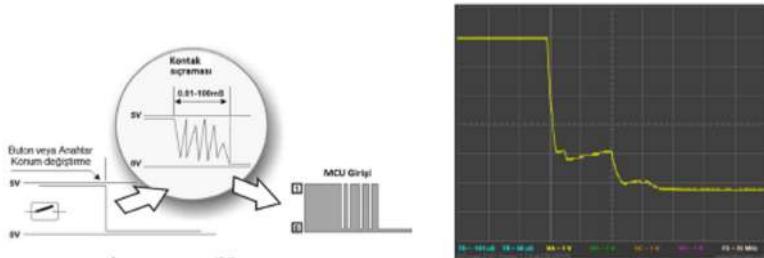


- PullUp bağlantıda GPIO girişi direnç üzerinden + beslemeye (VCC/VDD) bağlanır. Butona basılmadığı durumda GPIO girişinde lojik 1 vardır. Butona basıldığı durumda girişe 0V (lojik 0) uygulanmış olur.
- PullDown bağlantıda, GPIO girişi direnç üzerinden GND ye bağlanır. Butona basılmadığı durumda girişte lojik 0 bulunur. Butona basıldığı durumda buton üzerinden lojik 1 uygulanmış olur.





- Buton ve anahtarda konum değiştiğinde arktan dolayı mikrodenetleyici girişinde çok sayıda istenmeyen lojik değer oluşur. Bu duruma ark deniyor.



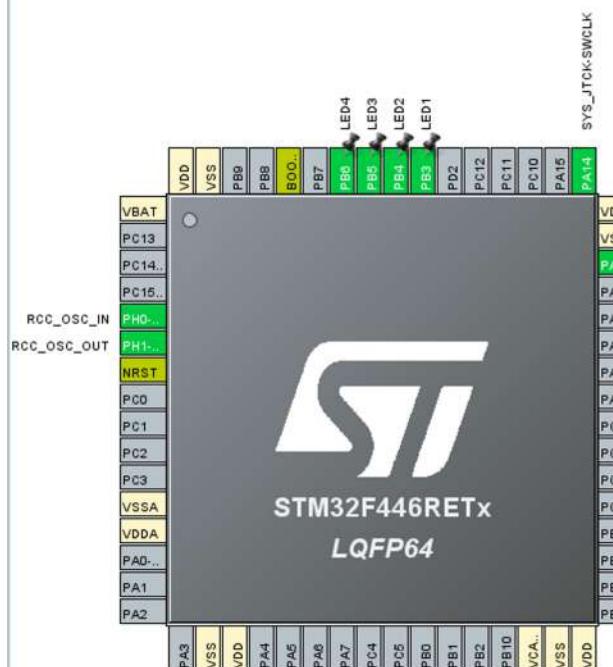
02_01 Harici Led Yakma

25 Aralık 2021 Cumartesi 00:52

02_01 Harici Led Yakma

➤ HAL

Konfigürasyon Kısmı



SYS_JTMS-SWDIO
SYS_JTCK-SWCLK

- System Core kısmından SYS tıklıyoruz ve Debug'da Serial Wire diyoruz.

Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum ou...	User Label	Modified
PA13	SYS_JTMS-SWDIO	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PA14	SYS_JTCK-SWCLK	n/a	n/a	n/a	n/a		<input type="checkbox"/>

- RCC tıklıyoruz HSE'de Crystal/Ceramic Resonator işaretliyoruz.

Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum ou...	User Label	Modified
PH0-OSC_IN	RCC_OSC_IN	n/a	n/a	n/a	n/a		<input type="checkbox"/>
PH1-OSC_OUT	RCC_OSC_OUT	n/a	n/a	n/a	n/a		<input type="checkbox"/>

- B portundaki 3., 4., 5. ve 6.pinlere led bağladık.

Pin Name	Signal ..	GPIO o...	GPIO mode	GPIO Pull-up/Pull-do...	Maximum output	User Label	Modified
PB3	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED1	<input checked="" type="checkbox"/>
PB4	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED2	<input checked="" type="checkbox"/>
PB5	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED3	<input checked="" type="checkbox"/>
PB6	n/a	Low	Output Push ...	No pull-up and no pul...	Very High	LED4	<input checked="" type="checkbox"/>

Kod Kısmı

- CubeMX'de ayarlamaları yaptıktan sonra main.c kısmında yaptığımız kısımları kendisi kod içerisinde oluşturmuştur.
- RCC ve GPIO için iki ayrı fonksiyon oluşturmuştur.
- Oluşturulan iki fonksiyon kod içerisinde yazmadan önce private function prototypes kısmında belirtilir.

```
48 /* Private function prototypes ----- */
49 void SystemClock_Config(void);
50 static void MX_GPIO_Init(void);
```

- Daha sonra bu iki fonksiyon kod içerisinde dahil edilir.

```

64@ int main(void)
65 {
66     /* USER CODE BEGIN 1 */
67
68     /* USER CODE END 1 */
69
70     /* MCU Configuration-----*/
71
72     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
73     HAL_Init();
74
75     /* USER CODE BEGIN Init */
76
77     /* USER CODE END Init */
78
79     /* Configure the system clock */
80     SystemClock_Config();
81
82     /* USER CODE BEGIN SysInit */
83
84     /* USER CODE END SysInit */
85
86     /* Initialize all configured peripherals */
87     MX_GPIO_Init();
88
89     /* USER CODE BEGIN 2 */
90
91     /* USER CODE END 2 */
92
93     /* Infinite loop */
94     /* USER CODE BEGIN WHILE */
95     while (1)
96     {
97         /* USER CODE END WHILE */
98
99         /* USER CODE BEGIN 3 */
100    }
101 }

```

- hal_gpio.c kısmından WritePin ve Toggle Pin fonksiyonlarına ulaşabiliriz.

```

410@ void HAL_GPIO_WritePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, GPIO_PinState PinState)
411 {
412     /* Check the parameters */
413     assert_param(IS_GPIO_PIN(GPIO_Pin));
414     assert_param(IS_GPIO_PIN_ACTION(PinState));
415
416     if(PinState != GPIO_PIN_RESET)
417     {
418         GPIOx->BSRR = GPIO_Pin;
419     }
420     else
421     {
422         GPIOx->BSRR = (uint32_t)GPIO_Pin << 16U;
423     }
424 }

433@ void HAL_GPIO_TogglePin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
434 {
435     uint32_t odr;
436
437     /* Check the parameters */
438     assert_param(IS_GPIO_PIN(GPIO_Pin));
439
440     /* get current Output Data Register value */
441     odr = GPIOx->ODR;
442
443     /* Set selected pins that were at low level, and reset ones that were high */
444     GPIOx->BSRR = ((odr & GPIO_Pin) << GPIO_NUMBER) | (~odr & GPIO_Pin);
445 }

```

- Döngümüze aşağıdaki kodu yazarız.

```

94     while (1)
95     {

```

```

94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99         HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin, GPIO_PIN_SET);
100        HAL_Delay(2000);
101        HAL_GPIO_TogglePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
102        HAL_Delay(500);
103    }

```

➤ SPL

Konfigürasyon Kısmı Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- Pinlerin nasıl kullanacağımızı seçtiğimiz kısımdır. Her iki bit bir pini temsil ediyor.

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Pini giriş, çıkış, analog ya da alternatif fonksiyon olarak mı kullanacağımızı belirtiyoruz. Alternatif fonksiyon ile kast edilen pinin çevresel birimlerden I2C, SPI olarak kullanılacağını belirtiyor.

Bits 2y:2y+1 MODERy[1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

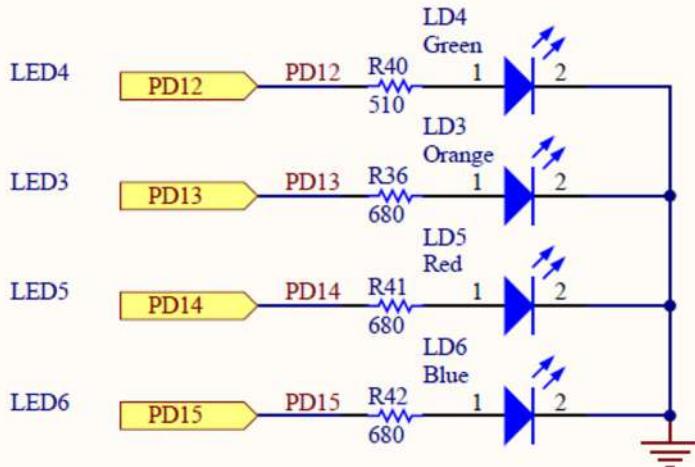
00: Input (reset state)

01: General purpose output mode

10: Alternate function mode

11: Analog mode

- STM32f407VG mikrodenetleyicisinin board üzerindeki led pinleri D12, D13, D14 ve D15'tir.



LEDs

- D portundaki belirlediğimiz pinlerimizi output yani çıkış olarak tanımlıyoruz ama öncesinde D portunu clock hattına aktarmamız gerekiyor.

```

GPIOD->MODER |= 1 << 24;           //PD12
GPIOD->MODER &= ~(1 << 25);
GPIOD->MODER |= 1 << 26;           //PD13
GPIOD->MODER &= ~(1 << 27);
GPIOD->MODER |= 1 << 28;           //PD14
GPIOD->MODER &= ~(1 << 29);
GPIOD->MODER |= 1 << 30;           //PD15
GPIOD->MODER &= ~(1 << 31);

```

- Output ayarı için her bir pine bir bit ayrılmıştır. Kullanacağımız push-pull reset durumunda 0 olduğundan bir ayar yapmamıza gerek yok.

GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 OTy: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

- Pinlerin very high speed olmasını istiyoruz. Pinleri 1 yapıyoruz.

GPIO port output speed register (GPIOx_OSPEEDR) (x = A..I/J/K)

Address offset: 0x08

Reset values:

- 0x0C00 0000 for port A
- 0x0000 00C0 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
OSPEEDR15 [1:0]		OSPEEDR14 [1:0]		OSPEEDR13 [1:0]		OSPEEDR12 [1:0]		OSPEEDR11 [1:0]		OSPEEDR10 [1:0]		OSPEEDR9 [1:0]		OSPEEDR8 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OSPEEDR7[1:0]		OSPEEDR6[1:0]		OSPEEDR5[1:0]		OSPEEDR4[1:0]		OSPEEDR3[1:0]		OSPEEDR2[1:0]		OSPEEDR1 [1:0]		OSPEEDR0 [1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 OSPEEDR[y:1:0]: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O output speed.

- 00: Low speed
- 01: Medium speed
- 10: High speed
- 11: Very high speed

Note: Refer to the product datasheets for the values of OSPEEDR[y] bits versus V_{DD} range and external load.

```
GPIOD->OSPEEDR |= 1 << 24;
GPIOD->OSPEEDR |= 1 << 25;
GPIOD->OSPEEDR |= 1 << 26;
GPIOD->OSPEEDR |= 1 << 27;
GPIOD->OSPEEDR |= 1 << 28;
GPIOD->OSPEEDR |= 1 << 29;
GPIOD->OSPEEDR |= 1 << 30;
GPIOD->OSPEEDR |= 1 << 31;
```

- Bu şekilde tek tek yazmak yerine 32 bitinin son sekize 1 yazılı hex kısmı yazabiliriz.

HEX	FF00 0000
DEC	-16.777.216
OCT	37 700 000 000
BIN	1111 1111 0000 0000 0000 0000 0000 0000
	WORD MS M+
0000 0000 0000 0000	
60 56 52 48	
0000 0000 0000 0000	
44 40 36 32	
1111 1111 0000 0000	
28 24 20 16	
0000 0000 0000 0000	
12 8 4 0	

GPIOD->OSPEEDR |= 0xFF000000;

- 8 biti 1 yaptığımızda HEX olarak FF veriri. 24.bite eklemeli şekilde de

yapabiliriz.

`GPIOD -> OSPEEDR |= FF << 24;`

- Kullanacağımız no pull-up, no pull-down reset durumunda 0 olduğundan bir ayar yapmamıza gerek yok.

GPIO port pull-up/pull-down register (**GPIOx_PUPDR**) (*x* = A..I/J/K)

Address offset: 0x0C

Reset values:

- 0x6400 0000 for port A
- 0x0000 0100 for port B
- 0x0000 0000 for other ports

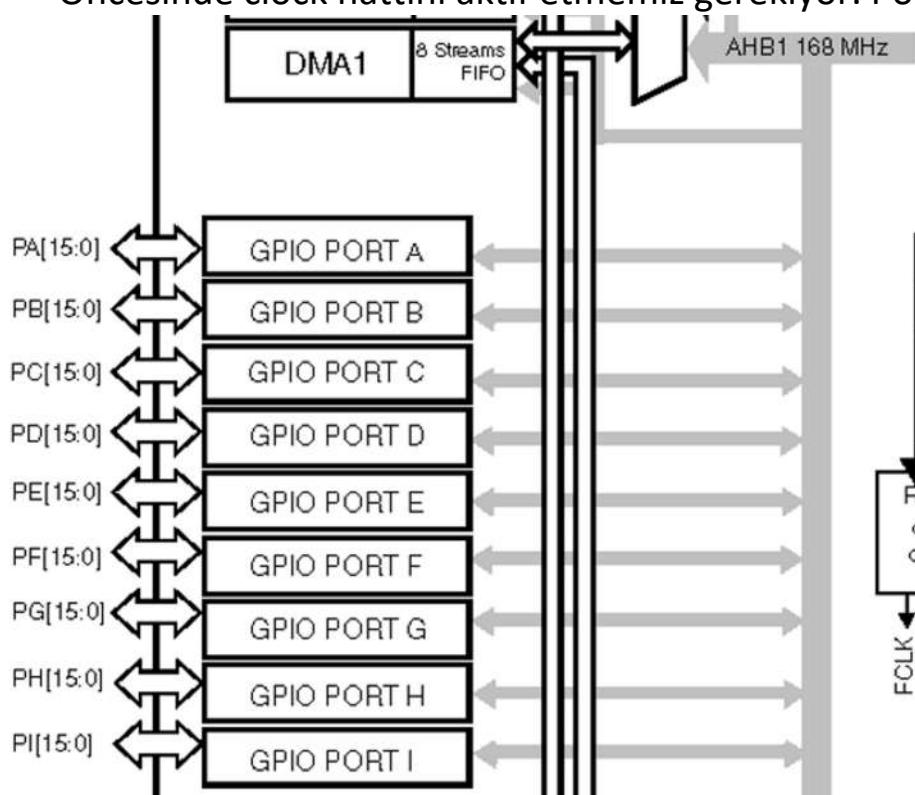
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PUPDR15[1:0]	PUPDR14[1:0]	PUPDR13[1:0]	PUPDR12[1:0]	PUPDR11[1:0]	PUPDR10[1:0]	PUPDR9[1:0]	PUPDR8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PUPDR7[1:0]	PUPDR6[1:0]	PUPDR5[1:0]	PUPDR4[1:0]	PUPDR3[1:0]	PUPDR2[1:0]	PUPDR1[1:0]	PUPDR0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **PUPDR_y[1:0]**: Port x configuration bits (*y* = 0..15)

These bits are written by software to configure the I/O pull-up or pull-down

- 00: No pull-up, pull-down
- 01: Pull-up
- 10: Pull-down
- 11: Reserved

- Öncesinde clock hattını aktif etmemiz gerekiyor. Portlar AHB1 kısmına gidiyor.



RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reser-ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved		DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved			
	rw	rw	rw	rw	rw	rw			rw	rw			rw				
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved			CRCE N	Reserved			GPIOIE N	GPIOH EN	GPIOG EN	GPIOFE N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	Reserved	
			rw				rw	rw	rw	rw	rw	rw	rw	rw			

- Biz D portunu kullandığımızdan sadece bunu aktif ediyoruz.

Bit 3 **GPIODEN**: IO port D clock enable

Set and cleared by software.

0: IO port D clock disabled

1: IO port D clock enabled

`RCC->AHB1ENR |= (1 << 3);`

- Pinleri set ya da reset edeceğimiz kısımdır. Bunları while döngüsü içerisinde yazıyoruz.

GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved																
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **ODRy**: Port output data (y = 0..15)

These bits can be read and written by software.

Note: For atomic bit set/reset, the ODR bits can be individually set and reset by writing to the GPIOx_BSRR register (x = A..I/J/K).

`GPIOD->ODR |= 1 << 12;`

`GPIOD->ODR |= 1 << 13;`

`GPIOD->ODR |= 1 << 14;`

`GPIOD->ODR |= 1 << 15;`

```

3⊕void RCC_Config(void)
4 {
5     RCC->CR &= ~(1 << 0);           //HSION
6     RCC->CR |= 1 << 16;             //HSEON
7     while(!(RCC->CR & (1 << 17))); //HSERDY
8     RCC->CR |= 1 << 19;             //CSSON
9     RCC->CFGR = 0x00000000;
10    RCC->PLLCFGR |= (1 << 22);      //PLLSRC
11    RCC->PLLCFGR |= (4 << 0);        //PLLM 4
12    RCC->PLLCFGR |= (168 << 6);       //PLLN 168
13    RCC->CR |= 1 << 24;              //PLLON
14    while(!(RCC->CR & (1 << 25))); //PLLRDY
15    RCC->CFGR &= ~(1 << 0);
16    RCC->CFGR |= (1 << 1);           //SW
17    while(!(RCC->CR & (1 << 1))); //SWS
18 }

20⊕void GPIO_Config(void)
21 {
22     RCC->AHB1ENR |= (1 << 3);        //D clock enable
23
24     //output mode
25     GPIOD->MODER |= 1 << 24;          //PD12
26     GPIOD->MODER &= ~(1 << 25);
27     GPIOD->MODER |= 1 << 26;          //PD13
28     GPIOD->MODER &= ~(1 << 27);
29     GPIOD->MODER |= 1 << 28;          //PD14
30     GPIOD->MODER &= ~(1 << 29);
31     GPIOD->MODER |= 1 << 30;          //PD15
32     GPIOD->MODER &= ~(1 << 31);
33
34     //GPIOD->OSPEEDR |= 1 << 24;
35     //GPIOD->OSPEEDR |= 1 << 25;
36     //GPIOD->OSPEEDR |= 1 << 26;
37     //GPIOD->OSPEEDR |= 1 << 27;
38     //GPIOD->OSPEEDR |= 1 << 28;
39     //GPIOD->OSPEEDR |= 1 << 29;
40     //GPIOD->OSPEEDR |= 1 << 30;
41     //GPIOD->OSPEEDR |= 1 << 31;
42     GPIOD->OSPEEDR |= 0xFF000000;      //Very high speed
43     //GPIOD->OSPEEDR |= FF << 24;
44 }

```

Kod Kısmı

- RCC ve GPIO için yazdığımız fonksiyonları ekledik ardından while döngüsü içersinde led blink uygulamasını gerçekleştirdik.

```
46 int main(void)
47 {
48     RCC_Config();
49     SystemCoreClockUpdate();
50
51     GPIO_Config();
52
53     while (1)
54     {
55         //set
56         GPIOD->ODR |= 1 << 12;
57         GPIOD->ODR |= 1 << 13;
58         GPIOD->ODR |= 1 << 14;
59         GPIOD->ODR |= 1 << 15;
60
61         for(int i=0; i<1680000; i++);
62
63         //reset
64         GPIOD->ODR |= ~(1 << 12);
65         GPIOD->ODR |= ~(1 << 13);
66         GPIOD->ODR |= ~(1 << 14);
67         GPIOD->ODR |= ~(1 << 15);
68
69         for(int i=0; i<1680000; i++);
70     }
71 }
```

02_02 Buton ile Led Yakma

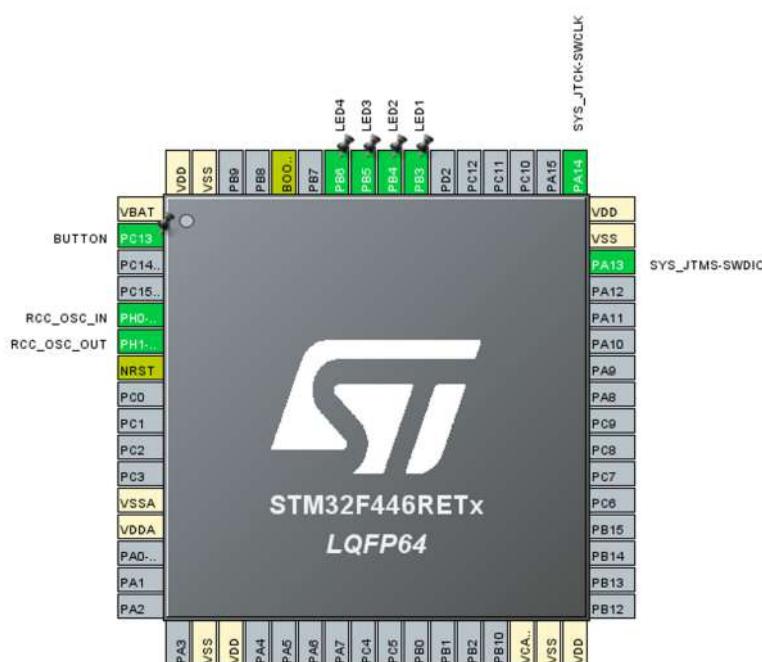
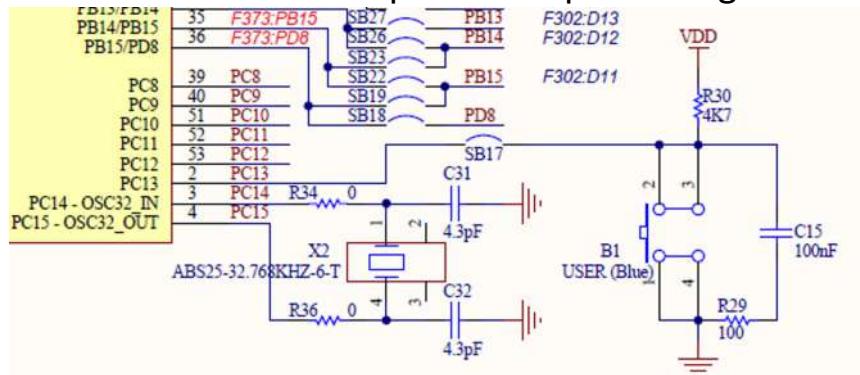
25 Aralık 2021 Cumartesi 00:52

02_02 Buton ile Led Yakma

➤ HAL

Konfigürasyon Kısmı

- Kullanıcı butonu C portun 13.pinine bağlı.



Pin Name	Signal on Pin	GPIO output..	GPIO mode	GPIO Pull-u...	Maximum o...	User Label	Modified
PB3	n/a	Low	Output Push...	No pull-up a...	Very High	LED1	✓
PB4	n/a	Low	Output Push...	No pull-up a...	Very High	LED2	✓
PB5	n/a	Low	Output Push...	No pull-up a...	Very High	LED3	✓
PB6	n/a	Low	Output Push...	No pull-up a...	Very High	LED4	✓
PC13	n/a	n/a	Input mode	Pull-down	n/a	BUTTON	✓

Kod Kısmı

- RCC için bir değişiklik yapmadığımızdan fonksiyon içeriği aynıdır.
- hal_gpio.c kısmından ReadPin fonksiyonlarına ulaşabiliriz.

```

375 GPIO_PinState HAL_GPIO_ReadPin(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin)
376 {
377     GPIO_PinState bitstatus;
378
379     /* Check the parameters */
380     assert_param(IS_GPIO_PIN(GPIO_Pin));
381
382     if((GPIOx->IDR & GPIO_Pin) != (uint32_t)GPIO_PIN_RESET)
383     {
384         bitstatus = GPIO_PIN_SET;
385     }
386     else
387     {
388         bitstatus = GPIO_PIN_RESET;
389     }
390     return bitstatus;
391 }

```

- Buton için count değişkeni atıyoruz.

```

44 /* USER CODE BEGIN PV */
45 int count=0;
46 /* USER CODE END PV */

```

- Okuma yaptığında yani butona bastığımızda if yapısının içine girer ve while ile kullanıcının eli butona basılı olup olmadığını kontrol ederiz. Elini çektiğinde count değerini 1 arttırıyor.
- Count değerinin modunu aldığımızda 0 iken RESET, 1 iken SET durumundadır.

```

94     while (1)
95     {
96         /* USER CODE END WHILE */
97
98         /* USER CODE BEGIN 3 */
99         if(HAL_GPIO_ReadPin(GPIOC, BUTTON_Pin))
100         {
101             while(HAL_GPIO_ReadPin(GPIOC, BUTTON_Pin));
102             HAL_Delay(100);
103             count++;
104         }
105
106         if(count % 2 == 1)
107         {
108             HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin,GPIO_PIN_SET);
109         }
110         else
111         {
112             HAL_GPIO_WritePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin,GPIO_PIN_RESET);
113         }
114     }

```

➤ SPL

Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- Öncelikle RCC ve GPIO ayarlarını yapıyoruz.
 - A portunu da kullanacağımdan A ve D portlarını RCC clocklarını aktif ediyoruz.
- RCC->AHB1ENR |= 0x00000009; //A, D clock enable
- Clock ayarlarını aşağıdaki gibi düzelttik.

- Önceki örneklerde yaptıklarımızın hepsini HEX formatında olacak şekilde düzelttik.
- Konunun videosu eski olduğundan önceki yaptığımız örneklerden farklılık olarak eskilerde RCC_CFG register kısmın 4:7 arası bitler koda eklenmişken yenilerde ekli olan 0:3 arası bitler eklenmemiş.
Eski videoda döngüde açılan flagların kapatılması için RCC_CIR registerin 11. ve 23.bitleri resetlemiştir.

```

10 void RCC_Config(void)
11 {
12     //RCC->CR &= ~(1 << 0);           //HSION
13     //RCC->CR |= 1 << 16;             //HSEON
14     RCC->CR |= 0x00030000;           //HSEON, HSERDY
15     //while(!(RCC->CR & (1 << 17)));
16     while(!(RCC->CR & 0x00020000)); //HSERDY
17     //RCC->CR |= 1 << 19;           //CSSON
18     RCC->CR |= 0x00080000;           //CSSON
19     RCC->CFG = 0x00000000;
20     //RCC->PLLCFGR |= (1 << 22);    //PLLSRC
21     RCC->PLLCFGR |= 0x00400000;      //PLLSRC
22     //RCC->PLLCFGR |= (4 << 0);      //PLLM 4
23     RCC->PLLCFGR |= 0x00000004;      //PLLM 4
24     //RCC->PLLCFGR |= (168 << 6);    //PLLN 168
25     RCC->PLLCFGR |= 0x00002A00;      //PLLN 168
26     RCC->PLLCFGR |= 0x00000000;      //PLLP 2
27     //RCC->CR |= 1 << 24;           //PLLON
28     RCC->CR |= 0x01000000;           //PLLON
29     //while(!(RCC->CR & (1 << 25)));
30     while(!(RCC->CR & 0x02000000)); //PLLRDY
31     //RCC->CFG &= ~(1 << 0);
32     //RCC->CFG |= (1 << 1);         //SW
33     RCC->CFG |= 0x00000001;          //SW
34     //while(!(RCC->CR & (1 << 1)));
35     while(!(RCC->CR & 0x00000001)); //SWS
36 }

```

- GPIO ayarlarını aşağıdaki gibi düzelttik.

```

38 void GPIO_Config(void)
39 {
40     //RCC->AHB1ENR |= (9 << 0);        //A, D clock enable
41     RCC->AHB1ENR |= 0x00000009;          //A, D clock enable
42
43     //GPIOD->MODER |= 1 << 24;         //PD12
44     //GPIOD->MODER &= ~(1 << 25);
45     //GPIOD->MODER |= 1 << 26;         //PD13
46     //GPIOD->MODER &= ~(1 << 27);
47     //GPIOD->MODER |= 1 << 28;         //PD14
48     //GPIOD->MODER &= ~(1 << 29);
49     //GPIOD->MODER |= 1 << 30;         //PD15
50     //GPIOD->MODER &= ~(1 << 31);
51     GPIOD->MODER |= 0x55000000;          //PD12, PD13, PD14, PD15
52     GPIOD->OTYPER |= 0x00000000;         //Output push-pull
53     GPIOD->OSPEEDR |= 0xFF000000;        //Very high speed
54     GPIOD->PUPDR |= 0x00000000;          //No pull-up, pull-down
55 }

```

Kod Kısmı

- Buton için A0 pini için okuma yapacağız. Burada ilk biti kullanacağız. While döngüsü içinde bir if yapısı içinde eğer okuma yapılrsa while döngüsüne girer ve okuma yani butona basma devam ediyorsa count değişkenini 1 arttırır.
- Buradaki delay ark olayından dolayı kullandık.

GPIO port input data register (GPIOx_IDR) (x = A..I/J/K)

Address offset: 0x10

Reset value: 0x0000 XXXX (where X means undefined)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IDR15	IDR14	IDR13	IDR12	IDR11	IDR10	IDR9	IDR8	IDR7	IDR6	IDR5	IDR4	IDR3	IDR2	IDR1	IDR0
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 IDRy: Port input data (y = 0..15)

These bits are read-only and can be accessed in word mode only. They contain the input value of the corresponding I/O port.

```
if(GPIOA->IDR & 0x00000001)
{
    while(GPIOA->IDR & 0x00000001);
    delay(1680000);

    count++;
}
```

- Count ve delay için globalde tanıttık.

```
3 int count = 0;
4
5 void delay(uint32_t time)
6 {
7     while(time--);
8 }
```

- Count değişkenini kalanı 0 ise yani 2.defa basıldığından ledi söndürken kalan 1 olduğunda yani 1.defa basıldığı durumda ledi yakacaktır.

```
57 int main(void)
58 {
59     RCC_Config();
60     SystemCoreClockUpdate();
61
62     GPIO_Config();
63
64     while (1)
65     {
66         if(GPIOA->IDR & 0x00000001)
67         {
68             while(GPIOA->IDR & 0x00000001);
69             delay(1680000);

70             count++;
71         }
72         if(count %2 == 0)
73             GPIOD->ODR |= 0x00000000;
74         else
75             GPIOD->ODR |= 0x0000F000;
76     }
77 }
78 }
```

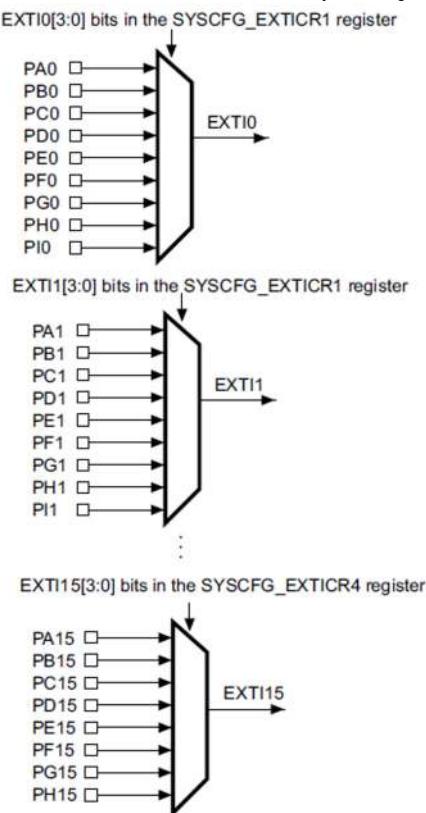
03 External Interrupt

5 Mayıs 2021 Çarşamba 08:02

03 External Interrupt

Giriş

- Önceliği yüksek işlerin mikrodenetleyici tarafından ana program akışını keserek yapılmasına interrupt (kesme) denir.
- Eğer bir kesme kaynağından mikrodenetleyiciye uyarı gelirse mikrodenetleyici yapmakta olduğu işi bekletir, kesme alt programına gider, o programı icra eder, daha sonra ana programda kaldığı yerden devam eder.
- Kesmeleri genellikle çok hızlı yapılması gereken işlemlerde, anlık tepki verilmesi gereken yerlerde kullanırız.
- Harici bir kaynaktan oluşan olaylardan dolayı meydana gelen kesmelere, harici kesmeler denir. Harici kaynak olarak, dış ortamdan pinler vasıtıyla gelecek kesme ve kandi içindeki donanımlardan gelen kesmeleri anlayabiliriz.
- STM32F407 mikrodenetleyicisi için porttaki 0.pin EXTI0 kanalına bağlıdır.



- Bunlar dışında 7 tane daha kanal vardır. Toplamda 23 kanal vardır.

EXTI line 16 is connected to the PVD output

EXTI line 17 is connected to the RTC Alarm event

EXTI line 18 is connected to the USB OTG FS Wakeup event

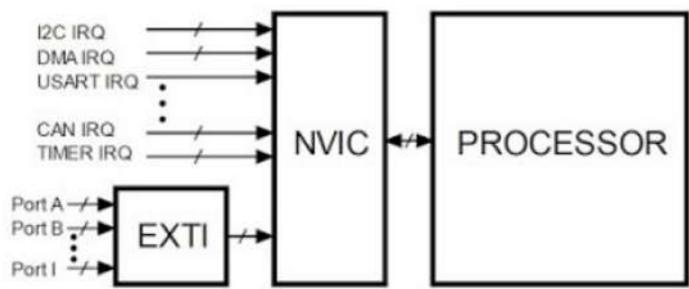
EXTI line 19 is connected to the Ethernet Wakeup event

EXTI line 20 is connected to the USB OTG HS (configured in FS) Wakeup event

EXTI line 21 is connected to the RTC Tamper and TimeStamp events

EXTI line 22 is connected to the RTC Wakeup event

- Karmaşık kesme isteklerinin işlemciye sürekli yük getirmemesi için işlemci içerisinde özel bir donanım bloğu oluşturulmuştur. Bu donanıma kesme kontrolörü (interrupt controller) adı verilir.
- Kesme kontrolörü haklı bir sebeple gelen kesme isteği neticesinde düzgün işleyen programı askıya alarak kesme fonksiyonu (interrupt function) olarak adlandırılan özel kod parçasını işlemeye başlar.
- Kesme fonksiyonunun işletilmesinin bitiminde program kaldığı yerden çalışmaya devam eder.
- NVIC kontrolör mikroişlemci içerisindeki önemli donanım kesmelerini (DMA istekleri, USART, CAN, I2C ve Timer gibi donanım kesmeleri) ve ayrıca External Interrupt/Event Controller (EXTI) adı verilen donanım vasıtıyla portlardan gelen kesmeleri kontrol eder.



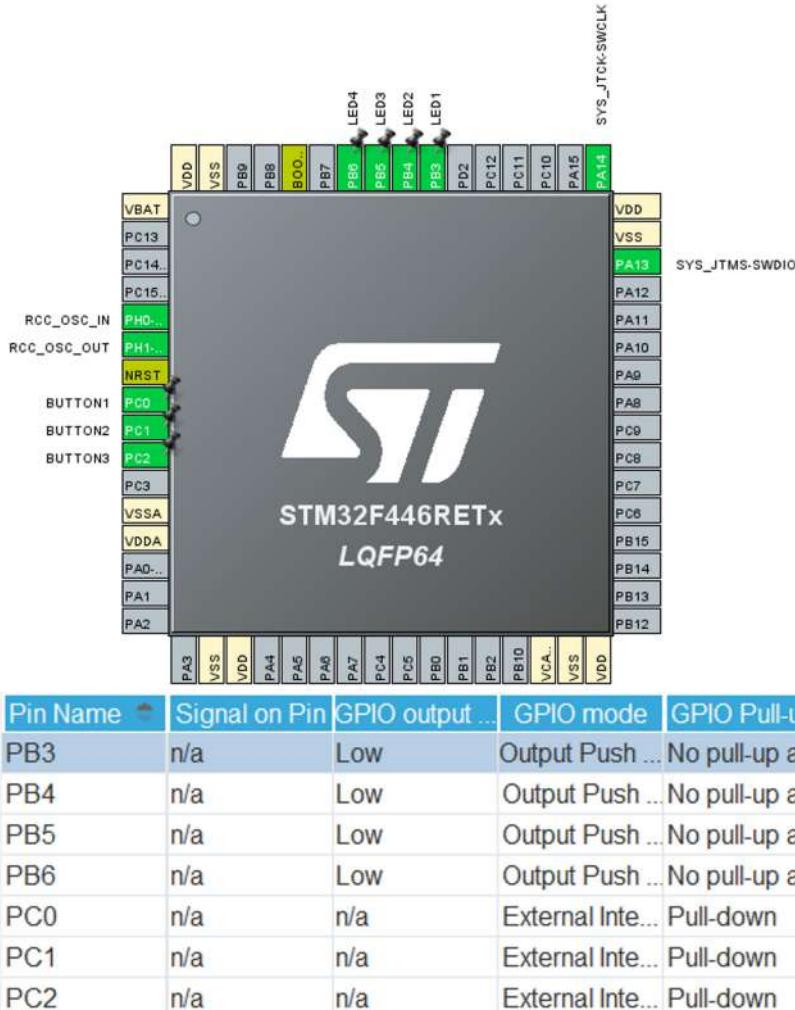
03_01 External Interrupt

25 Aralık 2021 Cumartesi 00:54

03_01 External Interrupt

➤ HAL

Konfigürasyon Kısmı



- NVIC kısmından işaretlediğimiz pinlerin kesme işlemi yapabilmesi için Enabled kısmı işaretliyoruz.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
EXTI line 0 interrupt	✓	0	0
EXTI line 1 interrupt	✓	0	0
EXTI line 2 interrupt	✓	0	0

Kod Kısmı

- it.c dosyasına gittiğimizde 3 adet kesme fonksiyonları oluşturulmuştur.

```
205 void EXTI0_IRQHandler(void)
206 {
207     /* USER CODE BEGIN EXTI0_IRQn 0 */
208
209     /* USER CODE END EXTI0_IRQn 0 */
210     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
211     /* USER CODE BEGIN EXTI0_IRQn 1 */
212
213     /* USER CODE END EXTI0_IRQn 1 */
214 }
```

```

219 void EXTI1_IRQHandler(void)
220 {
221     /* USER CODE BEGIN EXTI1_IRQn 0 */
222
223     /* USER CODE END EXTI1_IRQn 0 */
224     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
225     /* USER CODE BEGIN EXTI1_IRQn 1 */
226
227     /* USER CODE END EXTI1_IRQn 1 */
228 }

233 void EXTI2_IRQHandler(void)
234 {
235     /* USER CODE BEGIN EXTI2_IRQn 0 */
236
237     /* USER CODE END EXTI2_IRQn 0 */
238     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_2);
239     /* USER CODE BEGIN EXTI2_IRQn 1 */
240
241     /* USER CODE END EXTI2_IRQn 1 */
242 }

```

- İçerisinde yazılı olana Ctrl + Space tıkladığımızda hal_gpio.c dosyaya götürür. Burada en son satırda Callback fonksiyonunu çağırır.

```

492 void HAL_GPIO_EXTI_IRQHandler(uint16_t GPIO_Pin)
493 {
494     /* EXTI line interrupt detected */
495     if(__HAL_GPIO_EXTI_GET_IT(GPIO_Pin) != RESET)
496     {
497         __HAL_GPIO_EXTI_CLEAR_IT(GPIO_Pin);
498         HAL_GPIO_EXTI_Callback(GPIO_Pin);
499     }

```

- Buna aynı şekilde gittiğimizde bu dosya üzerinde yer alır. Biz main.c dosyamızda bu fonksiyonu kullanıp kesme işlemi yapacağız.

```

507 __weak void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
508 {
509     /* Prevent unused argument(s) compilation warning */
510     UNUSED(GPIO_Pin);
511     /* NOTE: This function Should not be modified, when the callback is needed,
512             the HAL_GPIO_EXTI_Callback could be implemented in the user file
513     */
514 }

```

- While döngüsü ile tüm pinleri yaktık ve count değişkeni değerinde delay süresi verdik.

```

23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25 int count=0 , i=0;
26 /* USER CODE END Includes */

125 while (1)
126 {
127     /* USER CODE END WHILE */
128
129     /* USER CODE BEGIN 3 */
130     HAL_GPIO_TogglePin(GPIOB, GPIO_PIN_All);
131     HAL_Delay(count);
132 }
133 /* USER CODE END 3 */
134 }

```

- Count değişkenini kesme olduğunda her buton için ayrı delay süreleri verdik.

```

55/* Private user code -----
56 /* USER CODE BEGIN 0 */
57void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
58 {
59     i++;
60     if(HAL_GPIO_ReadPin(GPIOC, BUTTON1_Pin))
61     {
62         if(i==1)
63         {
64             count=1000;
65         }
66         else if(i==2)
67         {
68             count=750;
69         }
70         else if(i==3)
71         {
72             count=500;
73         }
74         else
75         {
76             count=250;
77             i=0;
78         }
79     }
80     else if(HAL_GPIO_ReadPin(GPIOC, BUTTON2_Pin))
81     {
82         count=100;
83     }
84     else
85     {
86         count=50;
87     }
88 }
89 /* USER CODE END 0 */

```

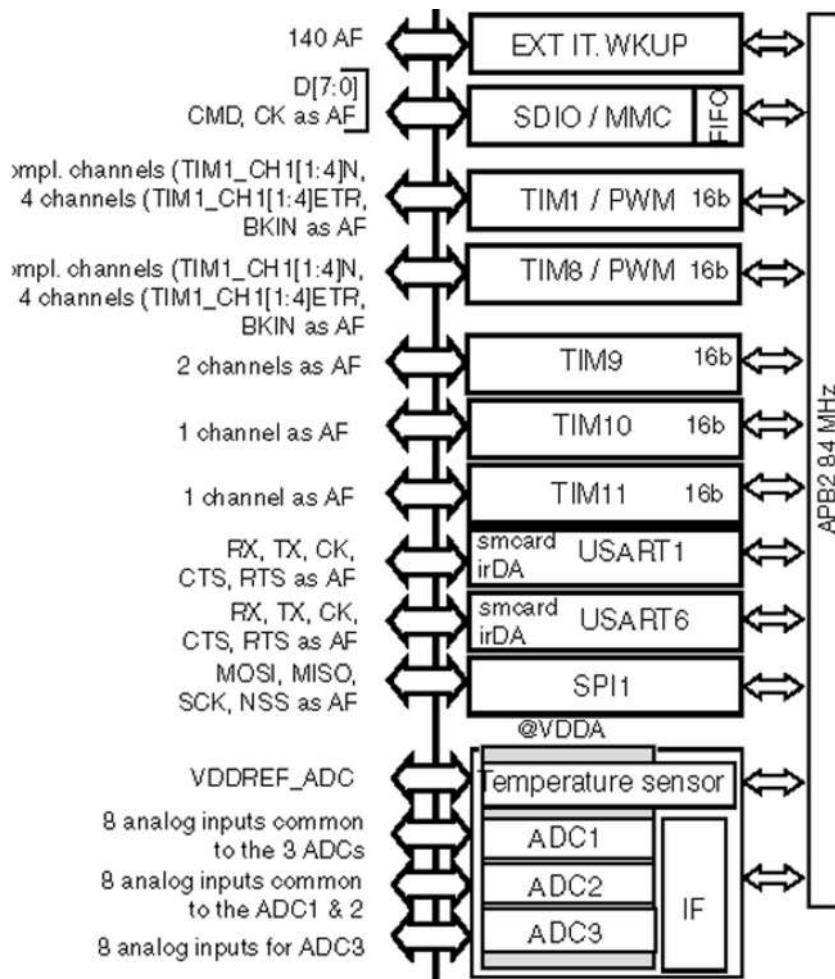
➤ SPL

Konfigürasyon Kısmı
Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- External Interrup'ın clock hattı APB2'ye gidiyor.



RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved														TIM11 EN	TIM10 EN	TIM9 EN
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reser- ved	SYSCF G GEN	Reser- ved	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reser- ved	USART 6 EN	USART 1 EN	Reser- ved	Tim8 EN	Tim1 EN			
	rw		rw	rw	rw	rw	rw		rw	rw		rw	rw			

Bit 14 **SYSCFGEN**: System configuration controller clock enable

Set and cleared by software.

0: System configuration controller clock disabled

1: System configuration controller clock enabled

- 14.pini 1 yapıyoruz.

RCC->APB2ENR |= 0x00004000; //SYSCGFEN

- 3 tane interrupt pini kullanacağımızdan öncelikle bunları aktif edip, öncelik sırası belirlemeliyiz.
- core_cm4.h kütüphanesinde bu fonksiyonları görebiliriz.

|1452| __STATIC_INLINE void NVIC_EnableIRQ(IRQn_Type IRQn)

- İkinci değere öncelik numarası yazıyoruz. Ne kadar küçük olursa öncelik sıralamasında önde olur.

```
1535@ __STATIC_INLINE void NVIC_SetPriority(IRQn_Type IRQn, uint32_t priority)
```

```
NVIC_EnableIRQ(EXTI0_IRQn);
NVIC_EnableIRQ(EXTI1_IRQn);
NVIC_EnableIRQ(EXTI2_IRQn);

NVIC_SetPriority(EXTI0_IRQn, 0);
NVIC_SetPriority(EXTI0_IRQn, 1);
NVIC_SetPriority(EXTI0_IRQn, 2);
```

- Interrupt olarak mı event olarak mı kullanacağımızı belirliyoruz. Interrupt olarak kullanacağız.

Interrupt mask register (EXTI_IMR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														MR22	MR21
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MR15	MR14	MR13	MR12	MR11	MR10	MR9	MR8	MR7	MR6	MR5	MR4	MR3	MR2	MR1	MR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 MR_x: Interrupt mask on line x

0: Interrupt request from line x is masked
1: Interrupt request from line x is not masked

- İlk 3 pine 1 yazıyoruz.

```
EXTI->EMR |= x00000007;
```

- Alçalan kenar mı yükselen kenar mı kullanacağımızı belirliyoruz. Yükselen kenar kullanacağız.

Rising trigger selection register (EXTI_RTSR)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														TR22	TR21
									rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
TR15	TR14	TR13	TR12	TR11	TR10	TR9	TR8	TR7	TR6	TR5	TR4	TR3	TR2	TR1	TR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 TR_x: Rising trigger event configuration bit of line x

0: Rising trigger disabled (for Event and Interrupt) for input line
1: Rising trigger enabled (for Event and Interrupt) for input line

İlk 3 pine 1 yazıyoruz.

```
EXTI->RTSR |= x00000007;
```

- Butonları external interrupt için aktif etmemiz gerekiyor.
Buton için A portunu kullanıyoruz.

SYSCFG external interrupt configuration register 1 (SYSCFG_EXTICR1)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EXTI3[3:0]				EXTI2[3:0]				EXTI1[3:0]				EXTI0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **EXTIx[3:0]**: EXTI x configuration (x = 0 to 3)

These bits are written by software to select the source input for the EXTIx external interrupt.

0000: PA[x] pin
0001: PB[x] pin
0010: PC[x] pin
0011: PD[x] pin
0100: PE[x] pin
0101: PF[x] pin
0110: PG[x] pin
0111: PH[x] pin
1000: PI[x] pin

SYSCFG->**EXTICR[0]** = 0x00000000;

- RCC, GPIO ve EXTI için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

8 void RCC_Config(void)
9 {
10    RCC->CR |= 0x00030000;           //HSEON, HSERDY
11    while(!(RCC->CR & 0x00020000)); //HSERDY
12    RCC->CR |= 0x00080000;         //CSSON
13    RCC->CFGR = 0x00000000;
14    RCC->PLLCFGR |= 0x00400000;   //PLLSRC
15    RCC->PLLCFGR |= 0x00000004;   //PLLM 4
16    RCC->PLLCFGR |= 0x00002A00;   //PLLN 168
17    RCC->PLLCFGR |= 0x00000000;   //PLLP 2
18    RCC->CR |= 0x01000000;        //PLLON
19    while(!(RCC->CR & 0x02000000)); //PLLRDY
20    RCC->CFGR |= 0x00000001;      //SW
21    while(!(RCC->CR & 0x00000001)); //SWS
22 }

24 void GPIO_Config(void)
25 {
26    RCC->AHB1ENR |= 0x00000009;    //A, D clock enable
27
28    GPIOD->MODER |= 0x55000000;   //PD12, PD13, PD14, PD15
29    GPIOD->OTYPER |= 0x00000000;  //Output push-pull
30    GPIOD->OSPEEDR |= 0xFF000000; //Very high speed
31    GPIOD->PUPDR |= 0x00000000;   //No pull-up, pull-down
32 }
```

```

34 void EXTI_Config(void)
35 {
36     RCC->APB2ENR |= 0x00004000;           //SYSCGFEN
37
38     NVIC_EnableIRQ(EXTI0_IRQn);
39     NVIC_EnableIRQ(EXTI1_IRQn);
40     NVIC_EnableIRQ(EXTI2_IRQn);
41
42     NVIC_SetPriority(EXTI0_IRQn, 0);
43     NVIC_SetPriority(EXTI0_IRQn, 1);
44     NVIC_SetPriority(EXTI0_IRQn, 2);
45
46     EXTI->EMR |= 0x00000007;
47     EXTI->RTSR |= 0x00000007;
48
49     SYSCFG->EXTICR[0] = 0x00000000;
50 }

```

Kod Kısmı

- İlk önce while döngüsünde tüm ledleri açıyoruz.

```

82 int main(void)
83 {
84     RCC_Config();
85     SystemCoreClockUpdate();
86     GPIO_Config();
87     EXTI_Config();
88
89     while (1)
90     {
91         GPIOD->ODR |= 0x0000F000;
92     }
93 }

```

- Pinde kesme olup olmadığını öğrenmemiz gerekiyor. Bitte değer olduğunda kesmeye girmış oluyor. Tekrar 1 yazarak bayrağı indiriyoruz.

Pending register (EXTI_PR)

Address offset: 0x14

Reset value: undefined

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved														PR22	PR21
									rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PR15	PR14	PR13	PR12	PR11	PR10	PR9	PR8	PR7	PR6	PR5	PR4	PR3	PR2	PR1	PR0
rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1	rc_w1

Bits 31:23 Reserved, must be kept at reset value.

Bits 22:0 PRx: Pending bit

0: No trigger request occurred

1: selected trigger request occurred

This bit is set when the selected edge event arrives on the external interrupt line.

This bit is cleared by programming it to '1'.

- Eğer kesmeye giriliyorsa 2 saniye boyunca belirlediğimiz ledi açıp diğerlerini kapatıyoruz.

```
51 void EXTI0_IRQHandler()
52 {
53     if(EXTI->PR & 0x00000001)
54     {
55         GPIOD->ODR |= 0x00001000;
56         delay(33600000);
57     }
58     EXTI->PR = 0x00000001;
59 }
```



```
61 void EXTI1_IRQHandler()
62 {
63     if(EXTI->PR & 0x00000002)
64     {
65         GPIOD->ODR |= 0x00002000;
66         delay(33600000);
67     }
68     EXTI->PR = 0x00000002;
69 }
```



```
71 void EXTI2_IRQHandler()
72 {
73     if(EXTI->PR & 0x00000004)
74     {
75         GPIOD->ODR |= 0x00004000;
76         delay(33600000);
77     }
78     EXTI->PR = 0x00000004;
79 }
```

04 ADC

5 Mayıs 2021 Çarşamba 08:02

04 ADC

Giriş

- Doğada var olan bütün fiziksel büyüklükler (ısı, ışık, ses, zaman vs.) analog büyüklik kavramına girer.
- Dünyadaki herhangi birşeyi dijital sistemlerimiz ile ölçmek, değerlendirmek, işlemek ve bu değerlere göre işlem yapabilmek için ADC (Analog Digital Converter) ihtiyaç vardır.
- ADC, analog sinyali kullandığımız dijital sistemimizin algılayabileceği dijital veri haline dönüştürür.
- ADC modülleri gerek harici, gerek dahili olsun hepsi bir referans voltajı ihtiyaç duyarlar.
- Genellikle mikroişlemcilerde referans voltajı işlemcinin besleme gerilimidir.
- Bu değer aynı zamanda ayarlar yapılarak harici olarak verilebilir.
- ADC'ler 10, 12, 16, 24 vb. bit çözünürlükte bulunurlar.
- STM32F4 discovery kartı üzerinde bulunan ADC ler 6,8,10 ve 12 bit çözünürlükte çalışabilirler.
- STM32F4 discovery kartında referans voltajı default 3.3V dur.
- ADC modülünün 10 bit olduğunu düşünelim. $2^{10} = 1024$ değeri okunacak maksimum değerdir.
- Yani ADC okuması yaparken $0V = 0$, $3.3V = 1023$ değeri ile bize döner.
- Buradan her bit bir değerinin alacağı voltaj değerini (çözünürlüğü) $3.3 / 1024 = 0,00322265625$ olarak buluruz.
- Buradan da biz ADC modülünden okuduğumuz değeri bu ifade ile çarparsak voltaj değerini buluruz. Örneğin 640 okuduysak $640 * 0,00322265625 = 2,0625$ V olduğunu buluruz.
- STM32F407VG işlemcisinde 3 adet ADC birimi mevcuttur.
- ADC biriminin ulaşabileceğİ maximum hız 36 MHz dir. Bu hız aynı zamanda ADC çözünürlüğü ile ters orantılıdır.
- ADC biriminin ölçüleceği en düşük gerilim değişimi (hassasiyet) çözünürlük denir.
- Çözünürlük arttıkça ADC biriminin ölçüm hızı düşmektedir.
- STM32F407VG mikrodenetleyicisinin en yüksek ADC çözünürlüğü 12 bittir.

CÖZÜNÜRLÜK

ADC ÇEVİRİM HIZI

12 Bit	12 Cycle
10 Bit	10 Cycle
8 Bit	8 Cycle
6 Bit	6 Cycle

- STM32F4 mikrodenetleyicisinde 0V-3.6V aralığında ölçümler yapılabilmektedir.
- Buradaki voltaj aralığında ADC biriminin beslemesi (Vdda-Vssa) ile ilgili bir durumdur.
- ADC biriminin besleme voltajı ve referans gerilimi (V_{ref+} , V_{ref-}) ADC biriminin ölçüleceği gerilim aralığını belirler.
- Her ne olursa olsun ADC birimi 3.6V dan fazlasını ölçemez.

- Analog bir değerden dijital bir değere dönüşüm yapılırken dikkat edilmesi gereken hususlar vardır.
- Bunlardan en önemlisi, ölçülecek analog gerilim ~~değerinin~~ dönüşümü yapacak çipin ölçüm aralığında olması gereklidir.
- Diğer önemli nokta, ölçüm yapılacak hassasiyetin belirlenmesi ve buna uygun bit genişliğinde bir dönüştürücü seçilmelidir.
- Ölçüm hassasiyetinde önemli olan dönüşüm yapacak sistemin bit çözünürlüğüdür(Resolution).
 - $\text{Resolution} = V_{ref+}/(2^n - 1)$
- Örneğin 8 bit çözünürlüğe sahip ve 0 - 3.3V aralığında arası ölçüm yapabilen bir ADC ölçüm ünitesinin ölçüleceği minimum değer (hassasiyet) yaklaşık olarak 12mV dur.
 - $\text{Resolution} = 3.3 / (2^8 - 1) = 3.3 / 255 = 0,01294\dots \approx 12\text{mV}$
- 12 bit çözünürlüğe sahip ve 0 - 3.3V aralığında arası ölçüm yapabilen bir ADC ölçüm ünitesinin ölçüleceği minimum değer (hassasiyet) yaklaşık olarak 805uV dur.
 - $\text{Resolution} = 3.3 / (2^{12} - 1) = 3.3 / 4095 \approx 805\text{uV}$

1- Tek Dönüşüm Modu: Sadece bir kere analog sinyali dijital sinyale çevirir.

2- Sürekli Dönüşüm Modu: Her dönüşüm işlemi bittiğinde, yeni bir dönüşüm işlemeye başlar. Yani sürekli olarak analog sinyalimizi dijital olarak çıktısını verir.

3- Tarama Modu: Bu mod ise birden fazla kanalda ADC çevrimi yaptığımız zaman, seçtiğimiz kanallarda tarama yaparak bu kanallarda dönüşüm yapmaya çalışır. Eğer bu modu sürekli mod gibi çalıştırırsak, seçtiğimiz kanalları sürekli olarak tarar. Aksi takdirde seçtiğimiz kanallar bir kere taranır ve durur.

4- Süreksiz Mod: Bu modda ise kanallar üzerinden çevrim sayınız 8 den az olmalıdır.

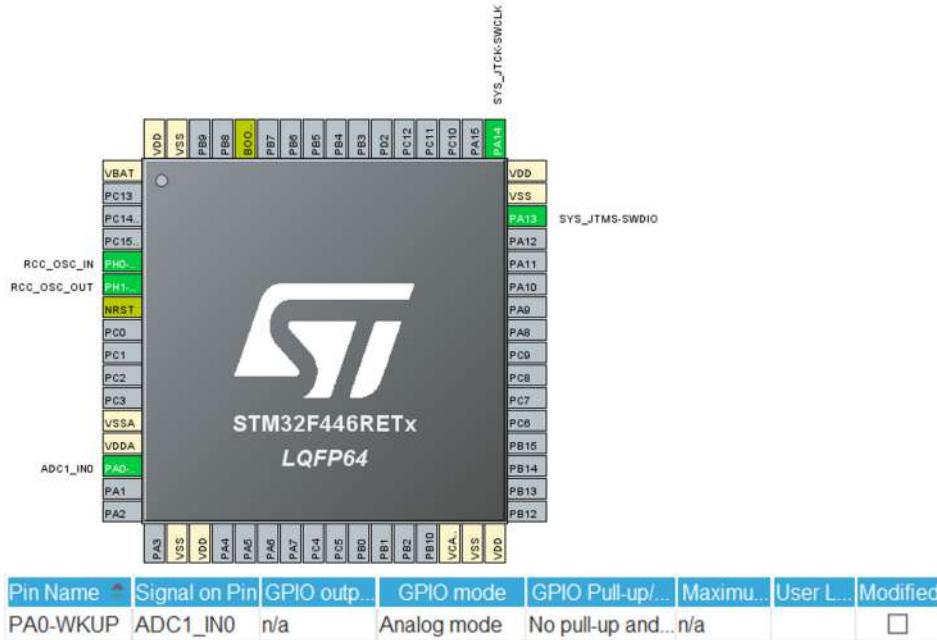
04_01 ADC Verisi Okuma

25 Aralık 2021 Cumartesi 00:56

04_01 ADC Verisi Okuma

➤ HAL

Konfigürasyon Kısmı



- Analog değer okumak için potansiyometreyi PA0'a bağladık. Bunun için Analog kısmından ADC1'den IN0 tıklarız. Gerçek voltaj değerini de hesaplayacağımızdan Vrefinit Channel'da seçilir.

- IN0
- IN1
- IN2
- IN3
- IN4
- IN5
- IN6
- IN7
- IN8
- IN9
- IN10
- IN11
- IN12
- IN13
- IN14
- IN15
- Temperature Sensor Channel

Vrefint Channel

- Daha sonra Parameter Settings'den Continuous Conversion Mode Enabled yapılır. Sürekli çevrim modu anlamına gelir ve tekrar tekrar okuma durumu yapar eğer çalışmazsa bir kere okur daha sonra değer okumaz.
- Çözünürlüğümüzü 12 bit yaptık yani en fazla 4095 değerini görebiliriz
- Çok kanal olduğundan Scan Mode Enabled yapılır.

ADC_Settings

Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled

ADC_Settings	
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel converts

- Number Of Conversion kısmı 2 kanal olduğundan iki yazılır ardından rank kısmından öncelik sırası verilir.
- Sampling Time ile kaç cycle'da bir çevrim yapacağımızı belirtiyoruz.

ADC-Regular_ConversionMode	
Number Of Conversion	2
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 0
Sampling Time	56 Cycles
Rank	2
Channel	Channel Vrefint
Sampling Time	56 Cycles

Kod Kısmı

- ADC okumada 3 mod var. Okumada Interrupt ve DMA kullanmayacağız. Polling Mode kullanımını yapacağız. Bunun nasıl kullanıldığını öğrenmek için hal_adc.c dosyasından bakıyoruz.

```

87    *** Polling mode IO operation ***
88    =====
89    [..]
90    (+) Start the ADC peripheral using HAL_ADC_Start()
91    (+) Wait for end of conversion using HAL_ADC_PollForConversion(), at this stage
92    user can specify the value of timeout according to his end application
93    (+) To read the ADC converted values, use the HAL_ADC_GetValue() function.
94    (+) Stop the ADC peripheral using HAL_ADC_Stop()

```

- ADC çevre birimini başlatmak için 717.satırındaki fonksiyon kullanılır.
- "*" var ise adresleme için "&" işaretini koymamız gerekiyor.

717@HAL_StatusTypeDef HAL_ADC_Start(ADC_HandleTypeDef* hadc)

- Çevrimin takibini 883.satırındaki fonksiyon ile yapıyoruz.
- Timeout için milisecond cinsinden istiyor.

883@HAL_StatusTypeDef HAL_ADC_PollForConversion(ADC_HandleTypeDef* hadc, uint32_t Timeout)

1566@uint32_t HAL_ADC_GetValue(ADC_HandleTypeDef* hadc)

840@HAL_StatusTypeDef HAL_ADC_Stop(ADC_HandleTypeDef* hadc)

- İşlemcinin beslemesi gerçekte 3.3V değildir. Bunun için ayrı hesap yapılması gereklidir. Hesap için 2 değer için birini ADC okumasından diğerini ise adres kısmından bulacağız.
- Adres kısmı için datasheet kısmından bakılması gereklidir.

Internal reference voltage calibration values

Symbol	Parameter	Memory address
V _{REFIN_CAL}	Raw data acquired at temperature of 30 °C VDDA = 3.3 V	0x1FFF 7A2A - 0x1FFF 7A2B

- Bu adres için kodda tanımlama yapılması gerekiyor.
- Tanımlama yaparken bizim kullanacağımız bit 12, adres biti 32 bittir. Bizim burada 32 britten 16 bite dönüştürme yapmamız gerekiyor.

```

23@/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #define VREFIN_CAL ((uint16_t*) ((uint32_t) 0x1FFF7A2A))
26
27 uint16_t adc_value[2];
28 float Vadc=0,Vdda=0;
29 int count=0;
30 /* USER CODE END Includes */

```

- İki kanalı adc_value değişkenine dizi halinde yazdık. 0.dizi rank'daki ayara göre ilk kanal oluyor.
- Diğer kanalı okuması için count değişkeni koyduk.

```

60/* Private user code -----
61 /* USER CODE BEGIN 0 */
62 void Read_ADC()
63 {
64     HAL_ADC_Start(&hadc1);
65
66     if(HAL_ADC_PollForConversion(&hadc1, 100000) == HAL_OK)
67     {
68         adc_value[count] = HAL_ADC_GetValue(&hadc1);
69         count++;
70
71         if(count==2)
72         {
73             count=0;
74         }
75
76         //Vadc=3.3*adc_value/4095;
77
78         Vdda=3.3>(*VREFIN_CAL)/adc_value[1];
79         Vadc=Vdda*adc_value[0]/4095;
80     }
81     HAL_ADC_Stop(&hadc1);
82 }
83 /* USER CODE END 0 */

```

Variable Name	Address/Expression	Read Value
Vadc	0x20000028	1.7121019
Vdda	0x2000002c	2.9433491
count	0x20000030	0

➤ SPL

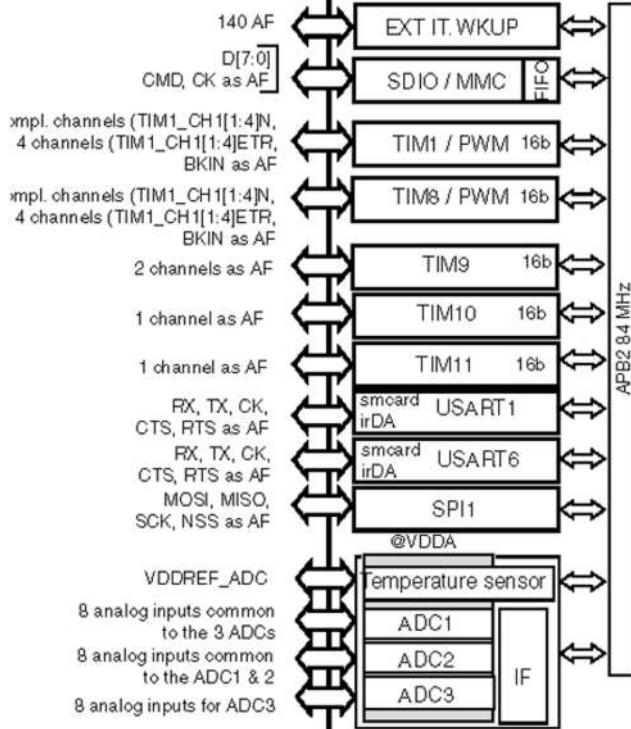
Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- ADC'nin clock hattı APB2'ye gitiyor.



RCC APB2 peripheral clock enable register (RCC_APB2ENR)

Address offset: 0x44

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reser- ved	SYSCF GEN	Reser- ved	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reser- ved	USART 6 EN	USART 1 EN	Reser- ved	TIM8 EN	TIM1 EN		
	rw		rw	rw	rw	rw	rw		rw	rw		rw	rw		

Bit 14 **SYSCFGEN**: System configuration controller clock enable

Set and cleared by software.

0: System configuration controller clock disabled
1: System configuration controller clock enabled

- 8.pini 1 yapıyoruz.

`RCC->APB2ENR |= 0x00000100;` //ADC1

ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0] JDISCE N DISC EN JAUTO AWDSG L SCAN JEOCIE AWDIE EOCIE AWDCH[4:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 25:24 **RES[1:0]**: Resolution

These bits are written by software to select the resolution of the conversion.

00: 12-bit (15 ADCCLK cycles)
01: 10-bit (13 ADCCLK cycles)
10: 8-bit (11 ADCCLK cycles)
11: 6-bit (9 ADCCLK cycles)

- 8 bit çözünürlük ayarlıyoruz.

`ADC1->CR1 |= 0x02000000;` //RES 8-bit

ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWST ART	EXTEN		EXTSEL[3:0]				reserved	JSWST ART	JEXTEN		JEXTSEL[3:0]			
	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON
				rw	rw	rw	rw							rw	rw

Bit 0 **ADON**: A/D Converter ON / OFF

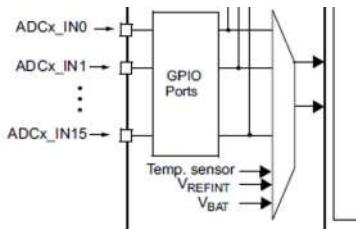
This bit is set and cleared by software.

Note: 0: Disable ADC conversion and go to power down mode
1: Enable ADC

- ADC aktif etmek için ilk bit 1 yaptı.

`ADC1->CR2 |= 0x00000001;` //ADON

- ADC_SMPMR register kısmında kaç cycle'da bir çevrim yapacağımızı belirtiyoruz.
- 0'dan 18'e kadar kanal var.



ADC sample time register 1 (ADC_SMPR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				SMP18[2:0]			SMP17[2:0]			SMP16[2:0]			SMP15[2:1]		
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP15_0	SMP14[2:0]			SMP13[2:0]			SMP12[2:0]			SMP11[2:0]			SMP10[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

ADC sample time register 2 (ADC_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				SMP9[2:0]			SMP8[2:0]			SMP7[2:0]			SMP6[2:0]		
				rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMP5_0	SMP4[2:0]			SMP3[2:0]			SMP2[2:0]			SMP1[2:0]			SMP0[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Biz 0.pinde çalıştığımız için 0.kanal kullanıyoruz.

Bits 31:30 Reserved, must be kept at reset value.

Bits 29:0 **SMPx[2:0]**: Channel x sampling time selection

These bits are written by software to select the sampling time individually for each channel.
During sample cycles, the channel selection bits must remain unchanged.

Note: 000: 3 cycles
001: 15 cycles
010: 28 cycles
011: 56 cycles
100: 84 cycles
101: 112 cycles
110: 144 cycles
111: 480 cycles

- 56 cycles'da çalışacağız.

ADC1->SMPR2 |= 0x00000003; //SMP0 56 cycles

ADC common control register (ADC_CCR)

Address offset: 0x04 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved				TSVREFE		VBATE	Reserved				ADCPRE				
				rw	rw	rw					rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DMA[1:0]	DDS	Res.	DELAY[3:0]				Reserved				MULTI[4:0]				
rw	rw	rw	rw	rw	rw	rw					rw	rw	rw	rw	

Bits 17:16 **ADCPRE**: ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADC. The clock is common for all the ADCs.

Note: 00: PCLK2 divided by 2
01: PCLK2 divided by 4
10: PCLK2 divided by 6
11: PCLK2 divided by 8

- ADC birimin maksimum clock hızı 36 MHz olması gerekiğinden CubeMX programında baktığımızda ADC birimin bağlı olduğu APB2 hattı 84MHz olduğundan bu clock hızını 4'e böldüğümüzde sağlamış oluyor. Bu sebeple 16. ve 17. bitleri ona göre düzenliyoruz.

ADC->CCR |= 0x00010000;

- RCC, GPIO ve ADC için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

21 void GPIO_Config(void)
22 {
23     RCC->AHB1ENR |= 0x00000001;           //A clock enable
24
25     GPIOA->MODER |= 0x00000003;          //PA0 Analog mode
26     GPIOA->OSPEEDR |= 0xFF000000;        //Very high speed
27 }

5 void RCC_Config(void)
6 {
7     RCC->CR |= 0x00030000;                //HSEON, HSERDY
8     while(!(RCC->CR & 0x00020000));      //HSERDY
9     RCC->CR |= 0x00080000;                //CSSON
10    RCC->CFGR = 0x00000000;
11    RCC->PLLCFGR |= 0x00400000;          //PLLSRC
12    RCC->PLLCFGR |= 0x00000004;          //PLLM 4
13    RCC->PLLCFGR |= 0x00002A00;          //PLLN 168
14    RCC->PLLCFGR |= 0x00000000;          //PLLP 2
15    RCC->CR |= 0x01000000;                //PLLON
16    while(!(RCC->CR & 0x02000000));      //PLLRDY
17    RCC->CFGR |= 0x00000001;              //SW
18    while(!(RCC->CR & 0x00000001));      //SWS
19 }

29 void ADC_Config(void)
30 {
31     RCC->APB2ENR |= 0x00000100;          //ADC1
32
33     ADC1->CR1 |= 0x02000000;            //RES 8-bit
34     ADC1->CR2 |= 0x00000001;            //ADON
35     ADC1->SMPR2 |= 0x00000003;          //SMP0 56 cycles
36     ADC->CCR |= 0x00010000;             //ADCPRE 4
37 }

```

Kod Kısmı

- 8 bitlik değişken okuyacağımızdan ona göre değişken ataması yapıyoruz.
- 8 bitlik dönüş değerini bir ADC okuma fonksiyonu yazıyoruz ve bunu while döngüsünde adc_value değerini Read_ADC() fonksiyonunda tutmak için eşitliyoruz.
- Fonksiyonun içine değişken ataması yaptık.

```

40 uint8_t Read_ADC()
41 {
42     uint8_t value;
43
44 }

```

adc_value = Read_ADC();

ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved	SWST ART	EXTEN		EXTSEL[3:0]			reserved	JSWST ART	JEXTEN		JEXTSEL[3:0]				
	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved			ALIGN	EOCS	DDS	DMA	Reserved			CONT	ADON				
			rw	rw	rw	rw				rw	rw				

- ADC'nin yazılımsal olarak çevrimi başlatmak için 30.bit'i 1 yapıyoruz.

Bit 30 **SWSTART**: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of regular channels

Note: This bit can be set only when ADON = 1 otherwise no conversion is launched.

ADC1->CR2 |= 0x40000000; //SWSTART

- Çevrimin bitip bitmediğini ADC_SR registerinden öğreniyoruz.

ADC status register (ADC_SR)

Address offset: 0x00

ADC status register (ADC_SR)

Address offset: 0x00

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								OVR	STRT	JSTRT	JEOC	EOC	AWD	rc_w0	rc_w0

- 1.bit 1 oldu ise çevrim bittiği anlamına geliyor.

Bit 1 **EOC**: Regular channel end of conversion

This bit is set by hardware at the end of the conversion of a regular group of channels. It is cleared by software or by reading the ADC_DR register.

0: Conversion not complete (EOCS=0), or sequence of conversions not complete (EOCS=1)
1: Conversion complete (EOCS=0), or sequence of conversions complete (EOCS=1)

```
while(!(ADC1->SR & 0x00000002)); //EOC
```

- Daha sonra ADC_CR registerinden okuma yapıyoruz.

ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 48](#) and [Figure 49](#).

- Okunan değeri value değişkenine atıyoruz.

```
value=ADC1->DR;
```

```
40 uint8_t Read_ADC()
41 {
42     uint8_t value;
43
44     ADC1->CR2 |= 0x40000000;           //SWSTART
45     while(!(ADC1->SR & 0x00000002)); //EOC
46
47     value=ADC1->DR;                  //DATA
48     return value;
49 }
```



```
51 int main(void)
52 {
53     RCC_Config();
54     SystemCoreClockUpdate();
55     GPIO_Config();
56     ADC_Config();
57
58     while (1)
59     {
60         adc_value = Read_ADC();
61     }
62 }
```

- STMStudio programını açıp File kısmından Import variables tıklıyoruz ve çalışma dosyamızın debug içerisindeki elf. Uzantılı dosyayı seçiyoruz ve içerisinde okuyacağımız değişkeni import edip yandaki grafiğe ekliyoruz.
- A0'a bağladığımız potansiyometreyi çevirdikçe değer değişecektir.

Variable Name	Address/Expression	Read Value
adc_value	0x200000a8	253

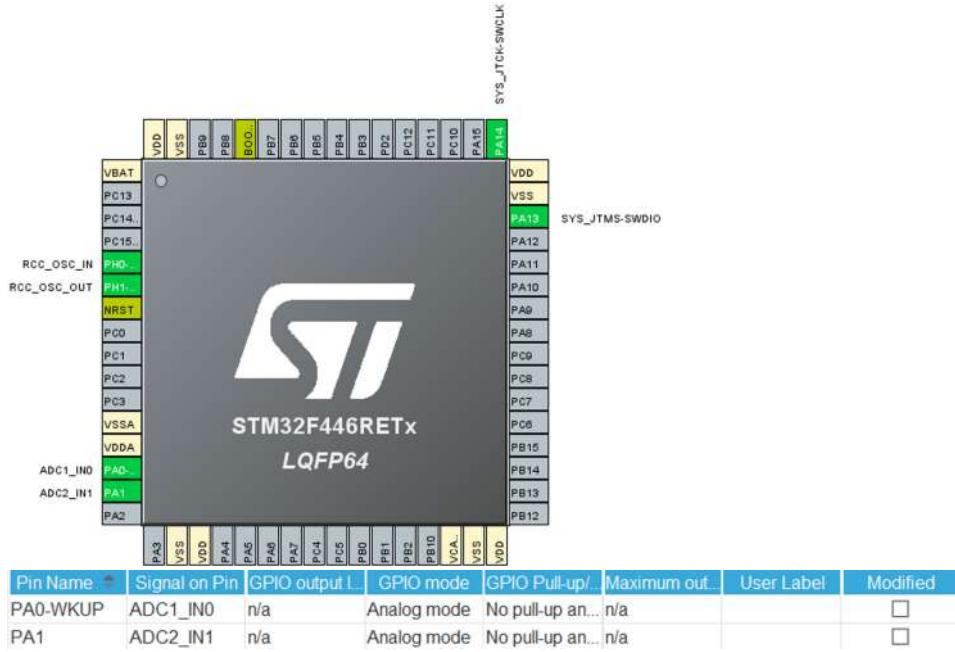
04_02 ADC Interrupt

25 Aralık 2021 Cumartesi 00:56

04_02 ADC Interrupt

➤ HAL

Konfigürasyon Kısımları



- ADC1 için IN0, Temperature Sensor Channel ve Vrefint Channel seçimi yaptık.

- IN0
 - IN1
 - IN2
 - IN3
 - IN4
 - IN5
 - IN6
 - IN7
 - IN8
 - IN9
 - IN10
 - IN11
 - IN12
 - IN13
 - IN14
 - IN15
- Temperature Sensor Channel
- Vrefint Channel

- Scan Mode ile Continuous Mode Enabled yapılır.

ADC_Settings	
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

- ADC1 için 3 kanal olduğundan her biri için rank işlemi yapılır.

ADC-Regular_ConversionMode	
Number Of Conversion	3
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 0
Sampling Time	56 Cycles
Rank	2
Channel	Channel Vrefint
Sampling Time	56 Cycles
Rank	3
Channel	Channel Temperature Sensor
Sampling Time	56 Cycles

- ADC2 için IN1 seçimi yapılır.
- ADC2 için tek kanal olduğundan Scan Mode seçimi yapılmaz.

ADC_Settings	
Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

ADC-Regular_ConversionMode	
Number Of Conversion	1
External Trigger Conversion Source	Regular Conversion launched by software
External Trigger Conversion Edge	None
Rank	1
Channel	Channel 1
Sampling Time	56 Cycles

- Interrupt kullanacağımızdan NVIC Settings kısmından Enabled yapılır.
- Sadece ADC1 değil ADC2 ve ADC3 için interrupts Enabled yapılmış olur.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
ADC1, ADC2 and ADC3 interrupts	<input checked="" type="checkbox"/>	0	0

Kod Kısmı

- ADC için Interrupt Mode kullanımı yapacağız.

```

96     *** Interrupt mode IO operation ***
97     =====
98     [...]
99     (+) Start the ADC peripheral using HAL_ADC_Start_IT()
100    (+) Use HAL_ADC_IRQHandler() called under ADC_IRQHandler() Interrupt subroutine
101    (+) At ADC end of conversion HAL_ADC_ConvCpltCallback() function is executed and user can
102        add his own code by customization of function pointer HAL_ADC_ConvCpltCallback
103    (+) In case of ADC Error, HAL_ADC_ErrorCallback() function is executed and user can
104        add his own code by customization of function pointer HAL_ADC_ErrorCallback
105    (+) Stop the ADC peripheral using HAL_ADC_Stop_IT()

```

```
1038= HAL_StatusTypeDef HAL_ADC_Start_IT(ADC_HandleTypeDef* hadc)
```

- Keseme olduğunda it.c dosyasındaki ADC_IRQHandler fonksiyonuna gelir ve fonksiyon içerisindeki HAL_ADC_IRQHandler fonksiyonunu çalıştırır.
- Bizim main.c dosyasında hazır olarak yazılmıştır.

```

206= void ADC_IRQHandler(void)
207 {
208     /* USER CODE BEGIN ADC_IRQHandler_0 */
209
210     /* USER CODE END ADC_IRQHandler_0 */
211     HAL_ADC_IRQHandler(&hadc1);
212     HAL_ADC_IRQHandler(&hadc2);
213     /* USER CODE BEGIN ADC_IRQHandler_1 */
214
215     /* USER CODE END ADC_IRQHandler_1 */
216 }

```

- 1578.satırındaki fonksiyonu main.c dosyasında kullanarak Interrupt'a girdiğinde çalışacak.

```
1578= __weak void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
```

- İçerisine önce hangi ADC'yi çalıştırıldığını öğrenmemiz gerekiyor. Bunun için _HAL_ADC_GET_FLAG kullanmamız gerekiyor.

129 (+) `_HAL_ADC_GET_FLAG: Get the selected ADC's flag status`

548 `#define __HAL_ADC_GET_FLAG(__HANDLE__, __FLAG__) (((__HANDLE__)->Instance->SR) & (__FLAG__)) == (__FLAG__)`

- Temperature için Datasheet kısmından aşağıdaki tabloları kullanıyoruz.

Temperature sensor characteristics

Symbol	Parameter	Min	Typ	Max	Unit
$T_L^{(1)}$	V_{SENSE} linearity with temperature	-	± 1	± 2	°C
Avg_Slope ⁽¹⁾	Average slope	-	2.5	-	mV/°C
$V_{25}^{(1)}$	Voltage at 25 °C	-	0.76	-	V
$t_{START}^{(2)}$	Startup time	-	6	10	μs
$T_{S_temp}^{(2)}$	ADC sampling time when reading the temperature (1 °C accuracy)	10	-	-	μs

Temperature sensor calibration values

Symbol	Parameter	Memory address
TS_CAL1	TS ADC raw data acquired at temperature of 30 °C, $V_{DDA}=3.3$ V	0x1FFF 7A2C - 0x1FFF 7A2D
TS_CAL2	TS ADC raw data acquired at temperature of 110 °C, $V_{DDA}=3.3$ V	0x1FFF 7A2E - 0x1FFF 7A2F

- Aşağıdaki formül üzerinden hesaplıyoruz.

$$\text{Temperature (in } ^\circ\text{C)} = \{(V_{SENSE} - V_{25}) / \text{Avg_Slope}\} + 25$$

- V_{25} = V_{SENSE} value for 25° C
- Avg_Slope = average slope of the temperature vs. V_{SENSE} curve (given in mV/°C or μ V/°C)

```

23/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #define VREFIN_CAL ((uint16_t*)((uint32_t)0x1FFF7A2A))
26 #define V25 (float) 0.76
27 #define Avg_Slope (float) 0.0025
28
29 uint16_t adc1_value[3], adc2_value;
30 float Vadc1, Vadc2, Vsense, Vdda, temperature;
31 int count=0;
32 /* USER CODE END Includes */

121 /* USER CODE BEGIN 2 */
122 HAL_ADC_Start_IT(&hadc1);
123 HAL_ADC_Start_IT(&hadc2);
124 /* USER CODE END 2 */

```

```

65/* Private user code -----*/
66 /* USER CODE BEGIN 0 */
67 void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
68 {
69     if(__HAL_ADC_GET_FLAG(&hadc1, ADC_FLAG_EOC) != RESET)
70     {
71         adc1_value[count]= HAL_ADC_GetValue(&hadc1);
72         count++; // count=0 => IN0, count=1 => Vrefint, count=2 => Temperature
73         if(count==3)
74         {
75             count=0;
76         }
77         Vdda= (float) 3.3 * (*VREFIN_CAL) / adc1_value[1];
78         Vadc1= Vdda * adc1_value[0] / 4095;
79
80         Vsense = Vdda * adc1_value[2] / 4095;
81         temperature= ((Vsense - V25) / Avg_Slope) + 25;
82     }
83     if(__HAL_ADC_GET_FLAG(&hadc2,ADC_FLAG_EOC) != RESET)
84     {
85         adc2_value=HAL_ADC_GetValue(&hadc2);
86         Vadc2= Vdda * adc2_value / 4095;
87     }
88 }
89 /* USER CODE END 0 */

```

Variable Name	Address/Expression	Read Value
Vadc1	0x20000028	2.9433491
Vadc2	0x2000002c	1.9205443
Vdda	0x20000030	2.9363744
Vsense	0x20000034	0.782018
count	0x20000038	0
temperature	0x2000003c	33.065987

05 DAC

5 Mayıs 2021 Çarşamba 08:02

05 DAC

Giriş

- DAC, dijital sinyali analog sinyale çevirme işlemi yapan birimdir.
- STM32F4 discovery kartında 0 - 3.3 V arasında tüm gerilimleri çıkış olarak vermemizi sağlar.
- STM32F407VG mikrodenetleyicisi içerisinde dahili olarak 12 bit tampona sahip, iki adet DAC birimi bulunur. Bu birimler sayesinde dijital bir veriyi analog bir veriye dönüştürerek çıkış üretilebilir.
- STM32F407VG'ye ait DAC birimleri 8 bit veya 12 bit değerinde çıkış üretilebilirler.
- 12 bit değerinde kullanılırken, veri 16 bitlik kaydedici içerisinde sola veya sağa dayalı şekilde kullanılabilir.
- DAC biriminin önemli özelliklerinden bir tanesi, gürültü veya sinyali üretebilme özelliğidir.
- Üçgen dalga üretebilme özelliğine sahiptir.
- Dönüşüm işlemi için harici tetikleyiciler kullanılabilir.
- DMA ile kullanılabilir.
- PA4 ve PA5 pinleri üzerinden çıkış verilmektedir.
- DAC birimleri APB1 veri yoluna bağlıdır, kullanmak için aktif etmek gereklidir.
- 12 bitlik çözünürlüğe sahip bir DAC biriminin referans gerilimleri $V_{SSA} = 0 \text{ V}$, $V_{DDA} = +3 \text{ V}$ ele alınır ise, adım başına üreteceği voltaj şu şekilde hesaplanır:
 - > $\text{DACout} = \text{Vref} / 4095$
- Buradan adım başına düşen voltaj:
 - > $\text{DACout} = 3V/4095 = 732,600732\dots$
- Buradan istenilen voltajı elde etmek için istenilen voltaj değeri adım başına üretilen voltaj değerine bölünerek elde edilir.
 - > Örneğin 1V elde etmek isteniyorsa:
 - $1/0,000732600\dots = 1365$ değeri elde edilir.

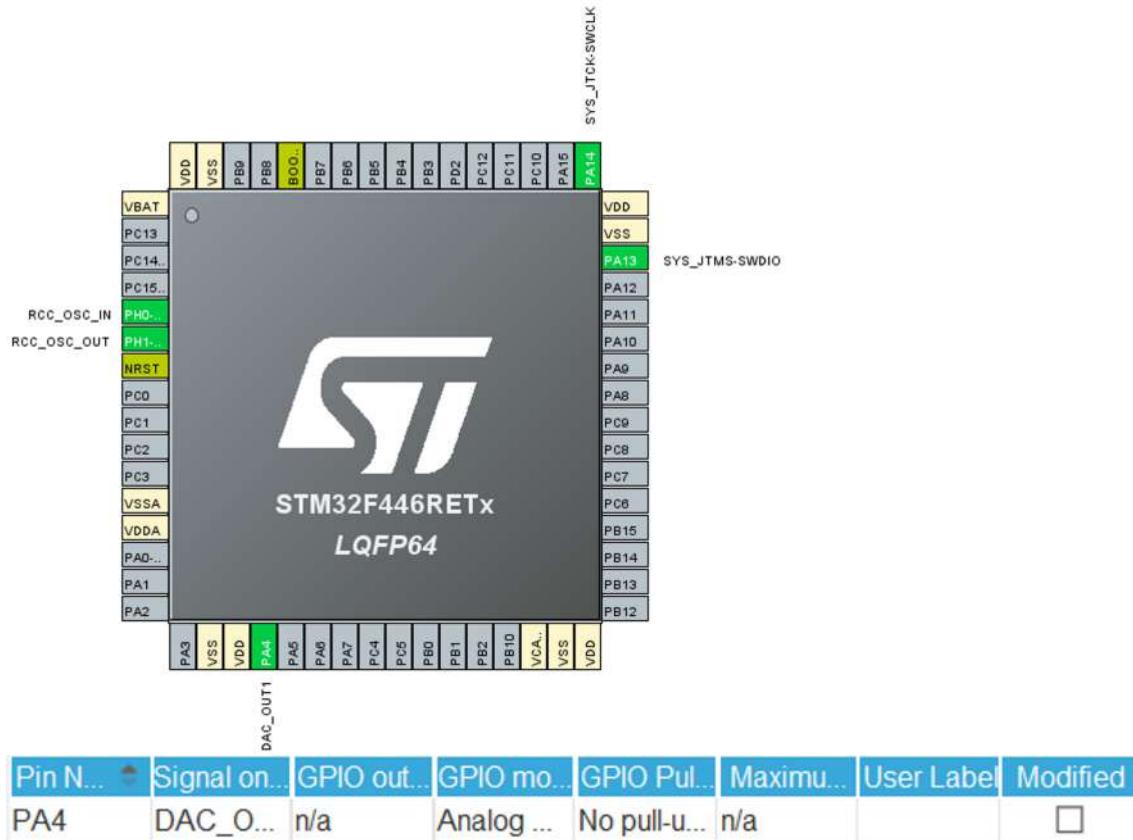
05_01 DAC Kullanımı

25 Aralık 2021 Cumartesi 00:57

05_01 DAC Kullanımı

► HAL

Konfigürasyon Kısmı



- DAC kısmından OUT1 seçiyoruz.

OUT1 Configuration

OUT2 Configuration

External Trigger

- Output Buffer, gürültü engelleme; Trigger tetiklemedir.
Biz bunları aynı şekilde bırakıyoruz.

DAC Out1 Settings

Output Buffer	Enable
Trigger	None

Kod Kısmı

- DAC için toplam iki kanal var.

```
18  [...]
19      *** DAC Channels ***
20      =====
21  [...]
22  STM32F4 devices integrate two 12-bit Digital Analog Converters
23
24  The 2 converters (i.e. channel1 & channel2)
25  can be used independently or simultaneously (dual mode):
26      (#) DAC channel1 with DAC_OUT1 (PA4) as output
27      (#) DAC channel2 with DAC_OUT2 (PA5) as output
```

- DAC kullanmak için iki modumuz var. Biri Polling diğeri DMA'dır.

- Biz Polling Mode kullanıyoruz.

```

110     *** Polling mode IO operation ***
111 =====
112 [..]
113     (+) Start the DAC peripheral using HAL_DAC_Start()
114     (+) To read the DAC last data output value, use the HAL_DAC_GetValue() function.
115     (+) Stop the DAC peripheral using HAL_DAC_Stop()

```

435 HAL_StatusTypeDef HAL_DAC_Start(DAC_HandleTypeDef *hdac, uint32_t Channel)

- DAC için değer verme yapacağımızdan GetValue değil SetValue kullanacağız. Çıkışın voltaj değeri ile oynamış oluyoruz.

```

759 /**
760  * @brief Set the specified data holding register value for DAC channel.
761  * @param hdac pointer to a DAC_HandleTypeDef structure that contains
762  *             the configuration information for the specified DAC.
763  * @param Channel The selected DAC channel.
764  *             This parameter can be one of the following values:
765  *             @arg DAC_CHANNEL_1: DAC Channel1 selected
766  *             @arg DAC_CHANNEL_2: DAC Channel2 selected
767  * @param Alignment Specifies the data alignment.
768  *             This parameter can be one of the following values:
769  *             @arg DAC_ALIGN_8B_R: 8bit right data alignment selected
770  *             @arg DAC_ALIGN_12B_L: 12bit left data alignment selected
771  *             @arg DAC_ALIGN_12B_R: 12bit right data alignment selected
772  * @param Data Data to be loaded in the selected data holding register.
773  * @retval HAL status
774 */
775 HAL_StatusTypeDef HAL_DAC_SetValue(DAC_HandleTypeDef *hdac, uint32_t Channel, uint32_t Alignment, uint32_t Data)

```

- Data kısmına 12 bit çalıştığımızdan 0-4095 arasında değer yazabiliriz.

```

23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25 int i=0;
26 /* USER CODE END Includes */

```

```

91 /* USER CODE BEGIN 2 */
92 HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
93 /* USER CODE END 2 */

```

```

95 /* Infinite loop */
96 /* USER CODE BEGIN WHILE */
97 while (1)
98 {
99     /* USER CODE END WHILE */
100
101    /* USER CODE BEGIN 3 */
102    do
103    {
104        HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, i);
105        HAL_Delay(1);
106        i++;
107    }
108    while(i<4096);
109    i=0;
110 }
111 /* USER CODE END 3 */

```

➤ SPL

Konfigürasyon Kısmı
Kod Kısmı

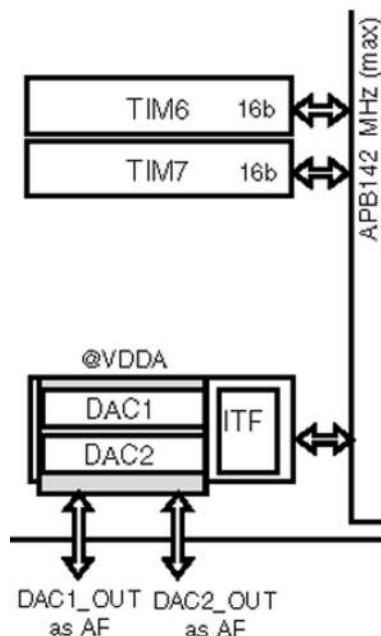
➤ REGISTER

Konfigürasyon Kısmı

- RCC için 5.satırda 0x00030000 yerine 0x00010000 yazdık.
HSERDY kısmı sadece okunabilir olduğundan o biti 1 yapmıyoruz.

```
3 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000; //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000; //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000; //PLLSRC
10    RCC->PLLCFGR |= 0x00000004; //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00; //PLLN 168
12    RCC->PLLCFGR |= 0x00000000; //PLLP 2
13    RCC->CR |= 0x01000000; //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001; //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17 }
```

- ADC'nin clock hattı APB1'e gidiyor.



7.3.13 RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	Reserved	CAN2 EN	CAN1 EN	Reserved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reserved
		rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved		WWDG EN	Reserved		TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN
rw	rw			rw			rw	rw	rw	rw	rw	rw	rw	rw	rw

- 29.pini 1 yapıyoruz.

Bit 29 **DACEN**: DAC interface clock enable

Set and cleared by software.

0: DAC interface clock disabled

1: DAC interface clock enable

```
RCC->APB1ENR |= 0x20000000; //DACE
```

DAC control register (DAC_CR)

Address offset: 0x00

Reset value: 0x0000 0000

- DAC işlemi için channel1 kullanacağımızdan 0.biti aktif etmemiz gerekiyor.

Bit 0 EN1: DAC channel1 enable

This bit is set and cleared by software to enable/disable DAC channel1.

0: DAC channel1 disabled

1: DAC channel1 enabled

```
DAC->CR |= 0x00000001;
```

- Yazılımsal tetikleme kullanmadığımızdan 0. biti 0 yapıyoruz.

DAC software trigger register (DAC_SWTRIGR)

Address offset: 0x04

Reset value: 0x0000 0000

Bit 0 SWTRIG1: DAC channel1 software trigger

This bit is set and cleared by software to enable/disable the software trigger.

0: Software trigger disabled

1: Software trigger enabled

Note: This bit is cleared by hardware (one APB1 clock cycle later) once the DAC_DHR1 register value has been loaded into the DAC_DOR1 register.

```
DAC->SWTRIGR |= 0x00000000;
```

- Channel1 için 12 bit sağa ya da sola veya 8 bit sağa kaydedebiliriz.

DAC channel1 12-bit right-aligned data holding register

(DAC_DHR12R1)

Address offset: 0x08

Reset value: 0x0000 0000

DAC channel1 12-bit left aligned data holding register (DAC_DHR12L1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[11:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

DAC channel1 8-bit right aligned data holding register (DAC_DHR8R1)

Address offset: 0x10

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DACC1DHR[7:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- 12 bit sağa kaydedeceğiz.

Bits 11:0 DACC1DHR[11:0]: DAC channel1 12-bit right-aligned data

These bits are written by software which specifies 12-bit data for DAC channel1.

RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reser- ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved			DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved	
	rw	rw	rw	rw	rw	rw				rw	rw			rw		
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved				CRCE N	Reserved			GPIOE N	GPIOH EN	GPIOG EN	GPIOF EN	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN
				rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

- DAC1 A4 pini olduğundan portunu sadece A portunu aktif ediyoruz.

RCC->[AHB1ENR](#) |= 0x00000001;

- RCC ve DAC1 için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

10 void RCC_Config(void)
11 {
12     RCC->CR |= 0x00010000; //HSEON
13     while(!(RCC->CR & 0x00020000)); //HSERDY
14     RCC->CR |= 0x00080000; //CSSON
15     RCC->CFGR = 0x00000000;
16     RCC->PLLCFGR |= 0x00400000; //PLLSRC
17     RCC->PLLCFGR |= 0x00000004; //PLLM 4
18     RCC->PLLCFGR |= 0x00002A00; //PLLN 168
19     RCC->PLLCFGR |= 0x00000000; //PLLP 2
20     RCC->CR |= 0x01000000; //PLLON
21     while(!(RCC->CR & 0x02000000)); //PLLRDY
22     RCC->CFGR |= 0x00000001; //SW
23     while(!(RCC->CR & 0x00000001)); //SWS
24 }

26 void DAC1_Config(void)
27 {
28     RCC->APB1ENR |= 0x20000000; //DACEEN
29     RCC->AHB1ENR |= 0x00000001; //A clock enable
30
31     DAC->CR |= 0x00000001; //DAC channel1 enabled
32     DAC->SWTRIGR |= 0x00000000; //Software trigger disabled
33     DAC->DHR12R1 |= 0x00000000; //DAC channel1 12-bit right-aligned data
34 }

```

Kod Kısmı

```

3 int i=0;
4
5 void delay(uint32_t time)
6 {
7     while(time--);
8 }

```

- Her i değeri arttığında i değeri kadar kaydetmiş oluyor.
- A4 pinine led bağladığımızda i değeri arttıkça voltaj değeri artacağından parlaklık artar.
- 4095 olduğunda yani 0x0000FFF olduğunda tam parlaklıktır yanar.

```

36 int main(void)
37 {
38     RCC_Config();
39     SystemCoreClockUpdate();
40     DAC1_Config();
41
42     while (1)
43     {
44         for(;i<4096; i++)
45         {
46             DAC->DHR12R1 |= i;
47             delay(16800);
48         }
49         i=0;
50     }
51 }

```

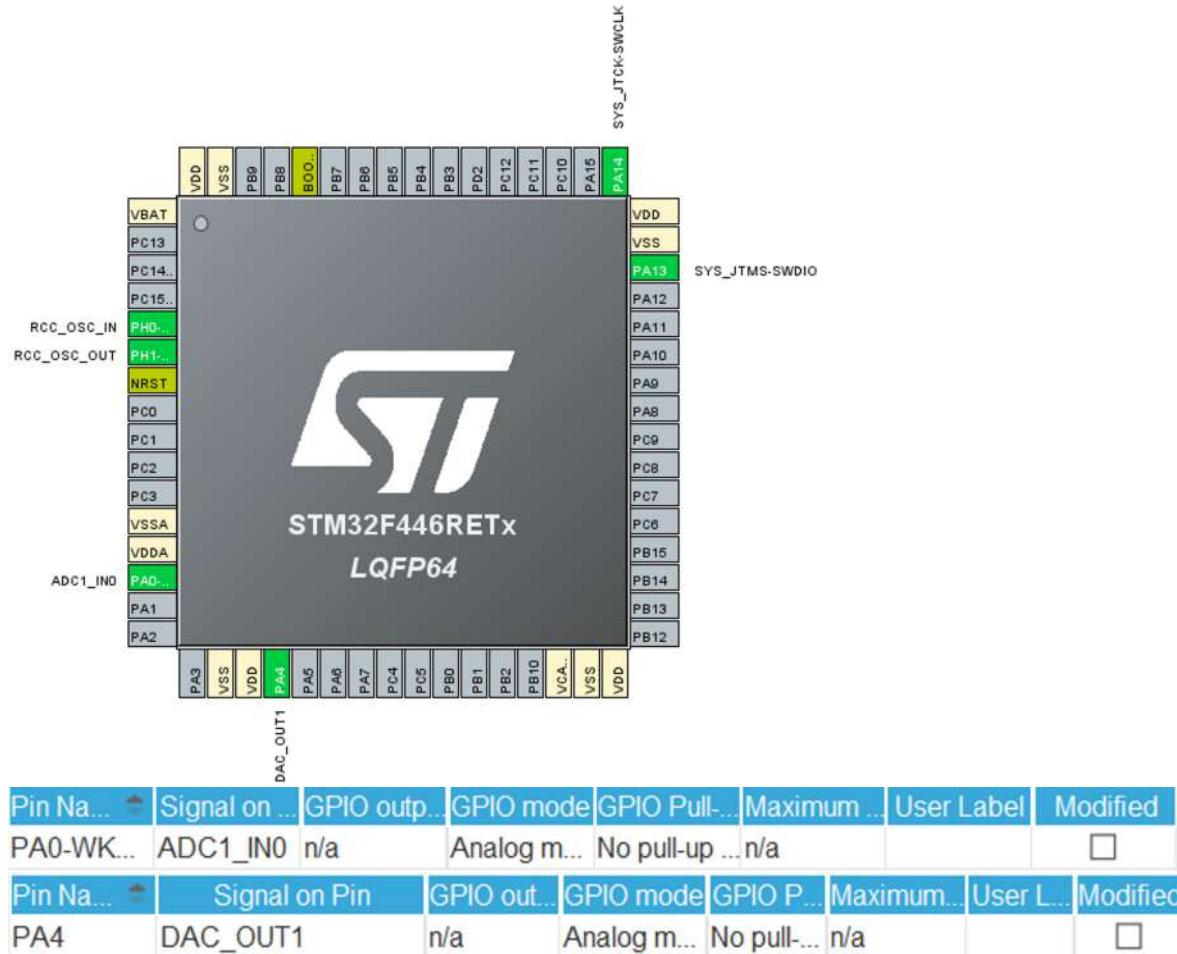
05_02 ADC Değeri ile DAC Kontrolü

25 Aralık 2021 Cumartesi 00:57

05_02 ADC Değeri ile DAC Kontrolü

HAL

Konfigürasyon Kısmı



- ADC1 için IN0, DAC için OUT1 seçilir.

ADC_Settings

Clock Prescaler	PCLK2 divided by 2
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

DAC Out1 Settings

Output Buffer	Enable
Trigger	None

Kod Kısmı

```
23/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 uint16_t adc_value, dac_value;
```

```

23/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 uint16_t adc_value, dac_value;
26 /* USER CODE END Includes */
• 16 bitlik okuma yapan fonksiyon yazdık.
• Geri dönüşlü fonksiyon olduğundan Return ile değeri geri
  döndürüyor.
60/* Private user code -----
61 /* USER CODE BEGIN 0 */
62uint16_t Read_ADC()
63 {
64     HAL_ADC_Start(&hadc1);
65
66     if(HAL_ADC_PollForConversion(&hadc1, 100000) == HAL_OK)
67     {
68         adc_value = HAL_ADC_GetValue(&hadc1);
69
70     }
71     HAL_ADC_Stop(&hadc1);
72
73     return adc_value;
74 }
75 /* USER CODE END 0 */

107 /* USER CODE BEGIN 2 */
108 HAL_DAC_Start(&hdac, DAC_CHANNEL_1);
109 /* USER CODE END 2 */

111 /* Infinite loop */
112 /* USER CODE BEGIN WHILE */
113 while (1)
114 {
115     /* USER CODE END WHILE */
116
117     /* USER CODE BEGIN 3 */
118     dac_value=Read_ADC();
119     HAL_DAC_SetValue(&hdac, DAC_CHANNEL_1, DAC_ALIGN_12B_R, dac_value);
120 }
121 /* USER CODE END 3 */
122 }

```

06 DMA

5 Mayıs 2021 Çarşamba 08:03

06 DMA

Giriş

- Çeşitli çevre birimlerinden okuduğumuz verileri bir değişkene atarız. Bu değişkenler RAM'de depolanır. Bu işlem normalde çevre birimlerinde okunan verinin CPU'ya alınıp ardından RAM'e yazılır. Ancak CPU kullanımını hem işlemciyi yorar hemde kayıplara yol açar. DMA (Doğrudan Bellek Erişimi) sayesinde verileri direk olarak RAM'e yazma imkanı buluruz.
- DMA donanımı CPU'dan bağımsız olarak verilerimizi peripheral (çevresel birim) den hafızaya, hafızadan çevresel birime ve hafızadan hafızaya olmak üzere hızlı bir şekilde kaynak adresinden hedef adrese aktarır.
 - > peripheral -> memory
 - > memory -> peripheral
 - > memory -> memory
- Bu sayede CPU'nun yükünü hafifletmiş oluruz. Sistem sanki 2 CPU ile çalışıyordu gibi düşünülebiliriz.
- Örneğin bilgisayarlarımıza bulunan 4 gerçek 4 sanal çekirdekteki sanal, aslında DMA diyebiliriz.
- DMA isteği için çevresel birim tarafından (ADC, DAC, I2c vs) DMA kontrolcüsüne istek gönderilir, kontrolcüde bu isteğin sırası gelince ilgili çevresel birime geri bildirimde bulunur ve işlem kaynak adresinden hedef adrese doğru gerçekleşir.
- STM32F4 ' te iki adet DMA vardır.
- DMA1'in DMA 2'den kanal 1'in kanal 2'den yüksek olduğu bilinmektedir.
 - > Öncelik sırası belirtmek için dört seviye vardır. Low, Medium, High, Very High.
- Aynı anda birçok kanal kullanıldığından hangi kanalın öncelik değeri fazla ise ilk o kanal alınır.
- DMA'lar paralel olarak çalışmazlar, seri olarak çalışırlar. Bu nedenle hangisinin sırası geldi ise o anda o çalışır.
- İlk olarak ADC den okuduğumuz değerleri, hiç CPU ya uğratmadan DMA ile direkt RAM'e yazacağız.
- Discovery boardda 2 adet DMA vardır ve ADC'yi DMA1 desteklemez, DMA2 destekler. Bu nedenle çalışma işlemi yaparken DMA2'yi kullanacağız.

06_01 DMA ile ADC Değer Okuma

25 Aralık 2021 Cumartesi 00:58

06_01 DMA ile ADC Değer Okuma

➤ HAL

Konfigürasyon Kısımlı

Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum output	User Label	Modified
PA0-WKUP	ADC1_IN0	n/a	Analog mode	No pull-up	n/a		<input type="checkbox"/>
PA1	ADC1_IN1	n/a	Analog mode	No pull-up	n/a		<input type="checkbox"/>

ADC_Settings

Clock Prescaler	PCLK2 divided by 4
Resolution	12 bits (15 ADC Clock cycles)
Data Alignment	Right alignment
Scan Conversion Mode	Enabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	EOC flag at the end of single channel conversion

- DMA ayarları için Add diyoruz. Ardından Stream, Direction ve Priority için seçenekleri seçiyoruz. Daha sonra Direction kısmından seçtiğimiz Peripheral ve Memory için adres değişikliği için Increment Adress için işaretliyoruz. Ardından gönderilecek data boyutu belirliyoruz. 12bit çalıştığımızdan 16 bitlik olan Half Word seçimi yapıyoruz.

DMA Request	Stream	Direction	Priority
ADC1	DMA2 Stream 0	Peripheral To Memory	Very High

DMA Request Settings

Mode	Circular	<input type="checkbox"/> Increment Address	<input checked="" type="checkbox"/> Memory
Use Fifo	<input type="checkbox"/>	Threshold	Half Word
		Data Width	Half Word
Burst Size			

Kod Kısmı

```

107      *** DMA mode IO operation ***
108      =====
109      [...]
110      (+) Start the ADC peripheral using HAL_ADC_Start_DMA(), at this stage the user specify the length
111          of data to be transferred at each end of conversion
112      (+) At The end of data transfer by HAL_ADC_ConvCpltCallback() function is executed and user can
113          add his own code by customization of function pointer HAL_ADC_ConvCpltCallback
114      (+) In case of transfer Error, HAL_ADC_ErrorCallback() function is executed and user can
115          add his own code by customization of function pointer HAL_ADC_ErrorCallback
116      (+) Stop the ADC peripheral using HAL_ADC_Stop_DMA()

```

- pData kısmına veriyi tutacağımız değişkeni yazıyoruz. Bizim değişken 16 bit olarak tanımladığımızdan dönüştürmemiz gerekiyor.
- Length ile kaç tane veri okuyacağımızı yazıyoruz.

```

1354 /**
1355  * @brief Enables ADC DMA request after last transfer (Single-ADC mode) and enables ADC
1356  * peripheral
1357  * @param hadc pointer to a ADC_HandleTypeDef structure that contains
1358  * the configuration information for the specified ADC.
1359  * @param pData The destination Buffer address.
1360  * @param Length The length of data to be transferred from ADC peripheral to memory.
1361  * @retval HAL status
1362 */
1363 HAL_StatusTypeDef HAL_ADC_Start_DMA(ADC_HandleTypeDef* hadc, uint32_t* pData, uint32_t Length)

46 /* USER CODE BEGIN PV */
47 uint16_t adc_value[2];
48 /* USER CODE END PV */

94 /* USER CODE BEGIN 2 */
95 HAL_ADC_Start_DMA(&hadc1, (uint32_t*)adc_value, 2);
96 /* USER CODE END 2 */

```

➤ SPL

Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- RCC için eklemeler yaptık.

RCC clock configuration register (RCC_CFGR)

Address offset: 0x08

Reset value: 0x0000 0000

Access: $0 \leq \text{wait state} \leq 2$, word, half-word and byte access

1 or 2 wait states inserted only if the access occurs during a clock source switch.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MCO2		MCO2 PRE[2:0]				MCO1 PRE[2:0]			I2SSCR	MCO1		RTCPRE[4:0]			
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PPRE2[2:0]			PPRE1[2:0]			Reserved	HPRE[3:0]				SWS1	SWS0	SW1	SW0	
rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	r	r	rw	rw	

- AHB, APB1 ve APB2 için clock hızını böldük.

Bits 7:4 **HPRE**: AHB prescaler

Set and cleared by software to control AHB clock division factor.

Caution: The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after HPRE write.

Caution: The AHB clock frequency must be at least 25 MHz when the Ethernet is used.

0xx: system clock not divided
 1000: system clock divided by 2
 1001: system clock divided by 4
 1010: system clock divided by 8
 1011: system clock divided by 16
 1100: system clock divided by 64
 1101: system clock divided by 128
 1110: system clock divided by 256
 1111: system clock divided by 512

RCC->**CFGREG** |= 0x00000000;

Bits 12:10 **PPRE1**: APB Low speed prescaler (APB1)

Set and cleared by software to control APB low-speed clock division factor.

Caution: The software has to set these bits correctly not to exceed 42 MHz on this domain.

The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE1 write.

0xx: AHB clock not divided
 100: AHB clock divided by 2
 101: AHB clock divided by 4
 110: AHB clock divided by 8
 111: AHB clock divided by 16

RCC->**CFGREG** |= 0x00001400;

Bits 15:13 **PPRE2**: APB high-speed prescaler (APB2)

Set and cleared by software to control APB high-speed clock division factor.

Caution: The software has to set these bits correctly not to exceed 84 MHz on this domain.

The clocks are divided with the new prescaler factor from 1 to 16 AHB cycles after PPRE2 write.

0xx: AHB clock not divided
 100: AHB clock divided by 2
 101: AHB clock divided by 4
 110: AHB clock divided by 8
 111: AHB clock divided by 16

RCC->**CFGREG** |= 0x00008000;

RCC clock interrupt register (RCC_CIR)

Address offset: 0x0C

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
								CSSC	Reserved	PLL12S RDYC	PLL RDYC	HSE RDYC	HSI RDYC	LSE RDYC	LSI RDYC
								W		W	W	W	W	W	W
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	PLL12S RDYIE	PLL RDYIE	HSE RDYIE	HSI RDYIE	LSE RDYIE	LSI RDYIE	CSSF	Reserved	PLL12S RDYF	PLL RDYF	HSE RDYF	HSI RDYF	LSE RDYF	LSI RDYF	
	rw	rw	rw	rw	rw	rw	r		r	r	r	r	r	r	r

- Kaldırılan bayrakları temizliyoruz.

Bit 19 **HSERDYC**: HSE ready interrupt clear

This bit is set by software to clear the HSERDYF flag.

0: No effect

1: HSERDYF cleared

RCC->CIR |= 0x00080000;

Bit 23 **CSSC**: Clock security system interrupt clear

This bit is set by software to clear the CSSF flag.

0: No effect

1: Clear CSSF flag

RCC->CIR |= 0x00800000;

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
3 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000; //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000; //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000; //PLLSRC
10    RCC->PLLCFGR |= 0x00000004; //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00; //PLLN 168
12    RCC->PLLCFGR |= 0x00000000; //PLLP 2
13    RCC->CR |= 0x01000000; //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001; //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000; //HPRE AHB 1
18    RCC->CFGR |= 0x00001400; //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000; //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000; //HSERDYC
21    RCC->CIR |= 0x00800000; //CSSC
22 }
```

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```
27 void GPIO_Config(void)
28 {
29     RCC->AHB1ENR |= 0x00000001; //A clock enable
30
31     GPIOA->MODER |= 0x00000003; //PA0 Analog mode
32     GPIOA->OSPEEDR |= 0x00000003; //Very high speed
33 }
```

ADC control register 1 (ADC_CR1)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved					OVRIE	RES		AWDEN	JAWDEN	Reserved					
					rw	rw	rw	rw	rw						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DISCNUM[2:0]			JDISCE N	DISC EN	JAUTO	AWDSG L	SCAN	JEOCIE	AWDIE	EOCIE	AWDCH[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Çoklu ADC okuması yapılacaksça scan yani tarama aktif edilmesi gereklidir.

Bit 8 **SCAN**: Scan mode

This bit is set and cleared by software to enable/disable the Scan mode. In Scan mode, the inputs selected through the ADC_SQRx or ADC_JSQRx registers are converted.

- 0: Scan mode disabled
1: Scan mode enabled

Note: An EOC interrupt is generated if the EOCIE bit is set:

- At the end of each regular group sequence if the EOCS bit is cleared to 0
- At the end of each regular channel conversion if the EOCS bit is set to 1

Note: A JEOC interrupt is generated only on the end of conversion of the last channel if the JEOCIE bit is set.

ADC1->**CR1** |= 1 << 8;

- Çözünürlük için 12-bit kullanıyoruz.

Bits 25:24 **RES[1:0]**: Resolution

These bits are written by software to select the resolution of the conversion.

- 00: 12-bit (15 ADCCLK cycles)
01: 10-bit (13 ADCCLK cycles)
10: 8-bit (11 ADCCLK cycles)
11: 6-bit (9 ADCCLK cycles)

ADC1->**CR1** |= 0 << 24;

ADC control register 2 (ADC_CR2)

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
reserved	SWST ART		EXTEN		EXTSEL[3:0]				reserved	JSWST ART		JEXTEN		JEXTSEL[3:0]		
	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
reserved				ALIGN	EOCS	DDS	DMA	Reserved						CONT	ADON	
				rw	rw	rw	rw							rw	rw	

- ADC aktif etmek için ilk bit 1 yapıldı.

Bit 0 **ADON**: A/D Converter ON / OFF

This bit is set and cleared by software.

Note: 0: Disable ADC conversion and go to power down mode
1: Enable ADC

- Sürekli okuma yapmasını istiyoruz.

Bit 1 **CONT**: Continuous conversion

This bit is set and cleared by software. If it is set, conversion takes place continuously until it is cleared.

- 0: Single conversion mode
1: Continuous conversion mode

- DMA mod ile çalışacağız.

Bit 8 DMA: Direct memory access mode (for single ADC mode)

This bit is set and cleared by software. Refer to the DMA controller chapter for more details.

0: DMA mode disabled

1: DMA mode enabled

- DMA'nın sürekli çalışması için 1 yapıyorum.

Bit 9 DDS: DMA disable selection (for single ADC mode)

This bit is set and cleared by software.

0: No new DMA request is issued after the last transfer (as configured in the DMA controller)

1: DMA requests are issued as long as data are converted and DMA=1

- 10.bit 1 yapılır.

Bit 10 EOCS: End of conversion selection

This bit is set and cleared by software.

0: The EOC bit is set at the end of each sequence of regular conversions. Overrun detection is enabled only if DMA=1.

1: The EOC bit is set at the end of each regular conversion. Overrun detection is enabled.

- Çevrim başlaması için 30.biti 1 yapıyoruz.

Bit 30 SWSTART: Start conversion of regular channels

This bit is set by software to start conversion and cleared by hardware as soon as the conversion starts.

0: Reset state

1: Starts conversion of regular channels

Note: This bit can be set only when ADON = 1 otherwise no conversion is launched

```
    Note: This bit can  
ADC1->CR2 |= 1 << 0;  
ADC1->CR2 |= 1 << 1;  
ADC1->CR2 |= 1 << 8;  
ADC1->CR2 |= 1 << 9;  
ADC1->CR2 |= 1 << 10;  
ADC1->CR2 |= 1 << 30;
```

- Kaç farklı kanaldan çevrim yapacağımızı belirtmemiz gerekiyor. Biz sadece A portun 0.bitinden yani 1 kanal'dan okuma yapacağım.

ADC regular sequence register 1 (ADC_SQR1)

Address offset: 0x2C

Reset value: 0x0000 0000

ADC regular sequence register 2 (ADC_SQR2)

Address offset: 0x30

Reset value: 0x0000 0000

ADC regular sequence register 3 (ADC_SQR3)

Address offset: 0x34

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved	SQ6[4:0]						SQ5[4:0]						SQ4[4:1]				
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
SQ4_0	SQ3[4:0]						SQ2[4:0]						SQ1[4:0]				
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- SQR1 için 20.biti 0 yaparak 1 tane kanal olduğunu belirtiyoruz.

Bits 23:20 L[3:0]: Regular channel sequence length

These bits are written by software to define the total number of conversions in the regular channel conversion sequence.

0000: 1 conversion

0001: 2 conversions

...

1111: 16 conversions

ADC1->**SQR1** |= 0 << 20;

- SQR3 için SQ1 kısmına kanal numarasını yazıyoruz.

Bits 4:0 **SQ1[4:0]**: 1st conversion in regular sequence

ADC1->**SQR3** |= 0 << 0;

ADC common control register (ADC_CCR)

Address offset: 0x04 (this offset address is relative to ADC1 base address + 0x300)

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved							TSVREFE		VBATE		Reserved				ADCPRE	
	rw	rw													rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
DMA[1:0]	DDS	Res.	DELAY[3:0]				Reserved				MULTI[4:0]					
rw	rw		rw	rw	rw	rw					rw	rw	rw	rw	rw	rw

Bits 17:16 **ADCPRE**: ADC prescaler

Set and cleared by software to select the frequency of the clock to the ADC. The clock is common for all the ADCs.

Note: 00: PCLK2 divided by 2

01: PCLK2 divided by 4

10: PCLK2 divided by 6

11: PCLK2 divided by 8

ADC->**CCR** |= 0x00010000;

- ADC için yazdığımız fonksiyon aşağıdaki gibidir.

```

32@void ADC_Config(void)
33 {
34     RCC->APB2ENR |= 0x00000100;           //ADC1
35
36     ADC1->CR1 |= 1 << 8;                //SCAN
37     ADC1->CR1 |= 0 << 24;               //RES 12-bit
38     ADC1->CR2 |= 1 << 0;                //ADON
39     ADC1->CR2 |= 1 << 1;               //CONT
40     ADC1->CR2 |= 1 << 8;                //DMA
41     ADC1->CR2 |= 1 << 9;               //DDS
42     ADC1->CR2 |= 1 << 10;              //EOCS
43     ADC1->CR2 |= 1 << 30;              //SWSTART
44     ADC1->SMPR2 |= 0x00000003;          //SMP0 56 cycles
45     ADC1->SQR1 |= 0 << 20;             //1 conversion
46     ADC1->SQR3 |= 0 << 0;             //SQ1
47     ADC->CCR |= 0x00010000.           //ADCPRE A

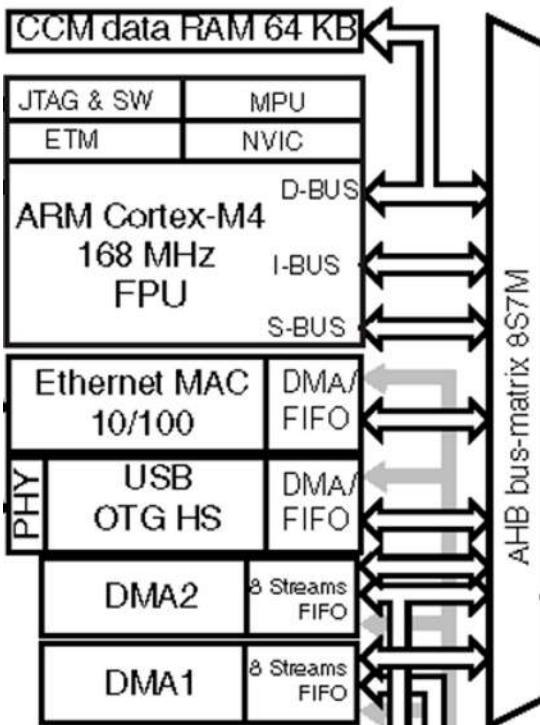
```

```

32 void ADC_Config(void)
33 {
34     RCC->APB2ENR |= 0x00000100;           //ADC1
35
36     ADC1->CR1 |= 1 << 8;             //SCAN
37     ADC1->CR1 |= 0 << 24;            //RES 12-bit
38     ADC1->CR2 |= 1 << 0;             //ADON
39     ADC1->CR2 |= 1 << 1;             //CONT
40     ADC1->CR2 |= 1 << 8;             //DMA
41     ADC1->CR2 |= 1 << 9;             //DDS
42     ADC1->CR2 |= 1 << 10;            //EOCS
43     ADC1->CR2 |= 1 << 30;            //SWSTART
44     ADC1->SMPR2 |= 0x00000003;        //SMP0 56 cycles
45     ADC1->SQR1 |= 0 << 20;            //1 conversion
46     ADC1->SQR3 |= 0 << 0;             //SQ1
47     ADC->CCR |= 0x00010000;           //ADCPRE 4
48 }

```

- DMA'nın clock hattı AHB'ye gidiyor.



RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser- ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved		DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved		
	rw	rw	rw	rw	rw	rw			rw	rw			rw			
Reserved		CRCE N		Reserved			GPIOE N	GPIOH EN	GPIOG EN	GPIOF N	GPIOEEN	GPIOD EN	GPIOC EN	GPIO BEN	GPIO AEN	
							rw	rw	rw	rw	rw	rw	rw	rw	rw	

Bit 22 DMA2EN: DMA2 clock enable

Set and cleared by software.

0: DMA2 clock disabled

1: DMA2 clock enabled

- 22.pini 1 yapıyoruz.

RCC->AHB1ENR |= 0x00400000;

- ADC için DMA2 kullanıyoruz. Tabloya baktığımızda biz burada Stream 4'ü seçiyoruz.

DMA1 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	SPI3_RX	-	SPI3_RX	SPI2_RX	SPI2_TX	SPI3_TX	-	SPI3_TX
Channel 1	I2C1_RX	-	TIM7_UP	-	TIM7_UP	I2C1_RX	I2C1_TX	I2C1_TX
Channel 2	TIM4_CH1	-	I2S3_EXT_RX	TIM4_CH2	I2S2_EXT_TX	I2S3_EXT_TX	TIM4_UP	TIM4_CH3
Channel 3	I2S3_EXT_RX	TIM2_UP TIM2_CH3	I2C3_RX	I2S2_EXT_RX	I2C3_TX	TIM2_CH1	TIM2_CH2 TIM2_CH4	TIM2_UP TIM2_CH4
Channel 4	UART5_RX	USART3_RX	UART4_RX	USART3_TX	UART4_TX	USART2_RX	USART2_TX	UART5_TX
Channel 5	UART8_TX ⁽¹⁾	UART7_TX ⁽¹⁾	TIM3_CH4 TIM3_UP	UART7_RX ⁽¹⁾	TIM3_CH1 TIM3_TRIG	TIM3_CH2	UART8_RX ⁽¹⁾	TIM3_CH3
Channel 6	TIM5_CH3 TIM5_UP	TIM5_CH4 TIM5_TRIG	TIM5_CH1	TIM5_CH4 TIM5_TRIG	TIM5_CH2	-	TIM5_UP	-
Channel 7	-	TIM6_UP	I2C2_RX	I2C2_RX	USART3_TX	DAC1	DAC2	I2C2_TX

1. These requests are available on STM32F42xxx and STM32F43xxx only.

DMA2 request mapping

Peripheral requests	Stream 0	Stream 1	Stream 2	Stream 3	Stream 4	Stream 5	Stream 6	Stream 7
Channel 0	ADC1	SAI1_A ⁽¹⁾	TIM8_CH1 TIM8_CH2 TIM8_CH3	SAI1_A ⁽¹⁾	ADC1	SAI1_B ⁽¹⁾	TIM1_CH1 TIM1_CH2 TIM1_CH3	-
Channel 1	-	DCMI	ADC2	ADC2	SAI1_B ⁽¹⁾	SPI6_TX ⁽¹⁾	SPI6_RX ⁽¹⁾	DCMI
Channel 2	ADC3	ADC3	-	SPI5_RX ⁽¹⁾	SPI5_TX ⁽¹⁾	CRYP_OUT	CRYP_IN	HASH_IN
Channel 3	SPI1_RX	-	SPI1_RX	SPI1_TX	-	SPI1_TX	-	-
Channel 4	SPI4_RX ⁽¹⁾	SPI4_TX ⁽¹⁾	USART1_RX	SDIO	-	USART1_RX	SDIO	USART1_TX
Channel 5	-	USART6_RX	USART6_RX	SPI4_RX ⁽¹⁾	SPI4_TX ⁽¹⁾	-	USART6_TX	USART6_TX
Channel 6	TIM1_TRIG	TIM1_CH1	TIM1_CH2	TIM1_CH1	TIM1_CH4 TIM1_TRIG TIM1_COM	TIM1_UP	TIM1_CH3	-
Channel 7	-	TIM8_UP	TIM8_CH1	TIM8_CH2	TIM8_CH3	SPI5_RX ⁽¹⁾	SPI5_TX ⁽¹⁾	TIM8_CH4 TIM8_TRIG TIM8_COM

1. These requests are available on STM32F42xxx and STM32F43xxx.

DMA stream x configuration register (DMA_SxCR) (x = 0..7)

This register is used to configure the concerned stream.

Address offset: $0x10 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

- 0.bit 0 olduğunda konfigürasyon yapmaya imkan tanıyor. Bunun için okuma yapacağız.

Bit 0 **EN**: Stream enable / flag stream ready when read low

This bit is set and cleared by software.

0: Stream disabled

1: Stream enabled

This bit may be cleared by hardware:

- on a DMA end of transfer (stream ready to be configured)
- if a transfer error occurs on the AHB master buses
- when the FIFO threshold on memory AHB port is not compatible with the size of the burst

When this bit is read as 0, the software is allowed to program the Configuration and FIFO bits registers. It is forbidden to write these registers when the EN bit is read as 1.

Note: Before setting EN bit to '1' to start a new transfer, the event flags corresponding to the stream in DMA_LISR or DMA_HISR register must be cleared.

- 1 olduğu sürece 0 olana kadar bekliyoruz. 0 olunca döngüden çıkış alt satırı geçecek.

```
while((DMA2_Stream4->CR & 0x00000001) == 1);
```

- ADC'den RAM'e yazacağımız bunun için Peripheral-to-memory seçiyoruz.

Bits 7:6 **DIR[1:0]**: Data transfer direction

These bits are set and cleared by software.

00: Peripheral-to-memory

01: Memory-to-peripheral

10: Memory-to-memory

11: reserved

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 0 << 6;
```

- Sürekli transfer için 8.biti 1 yapıyoruz.

Bit 8 **CIRC**: Circular mode

This bit is set and cleared by software and can be cleared by hardware.

0: Circular mode disabled

1: Circular mode enabled

When the peripheral is the flow controller (bit PFCTRL=1) and the stream is enabled (bit EN=1), then this bit is automatically forced by hardware to 0.

It is automatically forced by hardware to 1 if the DBM bit is set, as soon as the stream is enabled (bit EN ='1').

```
DMA2_Stream4->CR |= 1 << 8;
```

- Adresin değişmesini istemiyoruz. 9.biti 0 yapıyoruz.

Bit 9 **PINC**: Peripheral increment mode

This bit is set and cleared by software.

0: Peripheral address pointer is fixed

1: Peripheral address pointer is incremented after each data transfer (increment is done according to PSIZE)

This bit is protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 0 << 9;
```

Bit 10 **MINC**: Memory increment mode

This bit is set and cleared by software.

0: Memory address pointer is fixed

1: Memory address pointer is incremented after each data transfer (increment is done according to MSIZE)

This bit is protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 1 << 10;
```

- Peripheral ve Memory için data boyutunu 32 bit belirliyoruz.

Bits 12:11 **PSIZE[1:0]**: Peripheral data size

Bits 12:11 **PSIZE[1:0]**: Peripheral data size

These bits are set and cleared by software.

- 00: Byte (8-bit)
- 01: Half-word (16-bit)
- 10: Word (32-bit)
- 11: reserved

These bits are protected and can be written only if EN is '0'

Bits 14:13 **MSIZE[1:0]**: Memory data size

These bits are set and cleared by software.

- 00: byte (8-bit)
- 01: half-word (16-bit)
- 10: word (32-bit)
- 11: reserved

These bits are protected and can be written only if EN is '0'.

In direct mode, MSIZE is forced by hardware to the same value as PSIZE as soon as bit EN = '1'.

```
DMA2_Stream4->CR |= 2 << 11; //PSIZE Word (32-bit)  
DMA2_Stream4->CR |= 2 << 13; //MSIZE Word (32-bit)
```

- Önceliği çok yüksek yapıyoruz.

Bits 17:16 **PL[1:0]**: Priority level

These bits are set and cleared by software.

00: Low
01: Medium
10: High
11: Very high

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->CR |= 3 << 16;
```

- Kanal seçimi için 0.kanal yapıyoruz.

Bits 27:25 CHSEL[2:0]: Channel selection

These bits are set and cleared by software.

000: channel 0 selected
001: channel 1 selected
010: channel 2 selected
011: channel 3 selected
100: channel 4 selected
101: channel 5 selected
110: channel 6 selected
111: channel 7 selected

These bits are protected and can be written only if EN is '0'

- Kaç tane çevresel birimden yani ADC'den okuma yaptığımızı belirtiyoruz.

DMA stream x number of data register (DMA_SxNDTR) (x = 0..7)

Address offset: $0x14 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

Bits 15:0 **NDT[15:0]**: Number of data items to transfer

Number of data items to be transferred (0 up to 65535). This register can be written only when the stream is disabled. When the stream is enabled, this register is read-only, indicating the remaining data items to be transmitted. This register decrements after each DMA transfer.

Once the transfer has completed, this register can either stay at zero (when the stream is in normal mode) or be reloaded automatically with the previously programmed value in the following cases:

- when the stream is configured in Circular mode.
- when the stream is enabled again by setting EN bit to '1'

If the value of this register is zero, no transaction can be served even if the stream is enabled.

- Biz bir tane birimden yani ADC1'den okuma yapıyoruz.

`DMA2_Stream4->NDTR |= 1;`

- Okuma yaptığımız Çevresel birimin adresini yazıyoruz.

DMA stream x peripheral address register (DMA_SxPAR) (x = 0..7)

Address offset: $0x18 + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PAR[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PAR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **PAR[31:0]**: Peripheral address

Base address of the peripheral data register from/to which the data will be read/written.

These bits are write-protected and can be written only when bit EN = '0' in the DMA_SxCR register.

DMA transfer.

Once the transfer has completed, this register can either stay at zero (when the stream is in normal mode) or be reloaded automatically with the previously programmed value in the following cases:

- when the stream is configured in Circular mode.
- when the stream is enabled again by setting EN bit to '1'

If the value of this register is zero, no transaction can be served even if the stream is enabled.

ADC regular data register (ADC_DR)

Address offset: 0x4C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DATA[15:0]															
r	r	r	r	r	r	r	r	r	r	r	r	r	r	r	r

Bits 31:16 Reserved, must be kept at reset value.

Bits 15:0 **DATA[15:0]**: Regular data

These bits are read-only. They contain the conversion result from the regular channels. The data are left- or right-aligned as shown in [Figure 48](#) and [Figure 49](#).

`DMA2_Stream4->PAR |= (uint8_t) &ADC1->DR;`

- Değişkenin adresini yazıyoruz.

DMA stream x memory 0 address register (DMA_SxM0AR) (x = 0..7)

Address offset: $0x1C + 0x18 \times \text{stream number}$

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
M0A[31:16]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
M0A[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:0 **M0A[31:0]**: Memory 0 address

Base address of Memory area 0 from/to which the data will be read/written.

These bits are write-protected. They can be written only if:

- the stream is disabled (bit EN= '0' in the DMA_SxCR register) or
- the stream is enabled (EN='1' in DMA_SxCR register) and bit CT = '1' in the DMA_SxCR register (in Double buffer mode).

```
3 uint8_t adc;
4 uint8_t adc1[8];
```

```
DMA2_Stream4->M0AR |= (uint8_t) &adc1;
```

DMA stream x FIFO control register (DMA_SxFCR) (x = 0..7)

Address offset: $0x24 + 0x24 \times \text{stream number}$

Reset value: 0x0000 0021

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved																
FEIE								Reser ved		FS[2:0]			DMDIS		FTH[1:0]	
								r	r	r	r	r	rw	rw	rw	

Bits 1:0 **FTH[1:0]**: FIFO threshold selection

These bits are set and cleared by software.

- 00: 1/4 full FIFO
- 01: 1/2 full FIFO
- 10: 3/4 full FIFO
- 11: full FIFO

These bits are not used in the direct mode when the DMIS value is zero.

These bits are protected and can be written only if EN is '0'.

```
DMA2_Stream4->FCR |= 0 << 1;
```

- Stream4'ü fonksiyonun en son satırında aktif ediyoruz.

```
DMA2_Stream4->CR |= 1 << 0;
```

- Adres tanımları fonksiyonun ortalarında yazmak yerine while döngüsünün sonrasında yazdık.
- DMA için yazdığımız fonksiyon aşağıdaki gibidir.

```

53 void DMA_Config(void)
54 {
55     RCC->AHB1ENR |= 0x00400000;           //DMA2
56
57     while((DMA2_Stream4->CR & 0x00000001) == 1);
58     DMA2_Stream4->PAR |= (uint8_t) &ADC1->DR;          //PAR Peripheral address
59     DMA2_Stream4->M0AR |= (uint8_t) &adc1;             //M0A Memory 0 address
60     DMA2_Stream4->CR |= 0 << 6;                   //DIR
61     DMA2_Stream4->CR |= 1 << 8;                   //CIRC
62     DMA2_Stream4->CR |= 0 << 9;                   //PINC
63     DMA2_Stream4->CR |= 1 << 10;                  //PSIZE Word (32-bit)
64     DMA2_Stream4->CR |= 2 << 11;                  //MSIZE Word (32-bit)
65     DMA2_Stream4->CR |= 2 << 13;                  //PL
66     DMA2_Stream4->CR |= 3 << 16;                  //CHSEL
67     DMA2_Stream4->CR |= 0 << 25;                  //NDT
68     DMA2_Stream4->NDTR |= 1;                      //FTH 1/2 full FIFO
69     DMA2_Stream4->FCR |= 0 << 1;                 //EN Stream enabled
70     DMA2_Stream4->CR |= 1 << 0;
71 }

```

Kod Kısmı

- ADC için çevrimi DMA'dan sonra başlatmak için 80.satırı tekrar yazdık.

```

73 int main(void)
74 {
75     RCC_Config();
76     GPIO_Config();
77     ADC_Config();
78     DMA_Config();           //ADC1->CR2 |= ADC_CR2_SWSTART;
79     ADC1->CR2 |= 0x40000000;
80
81     while (1)
82     {
83
84     }
85 }
86

```

07 Timer

5 Mayıs 2021 Çarşamba 08:02

07 Timer

Giriş

- Timer modülünün temel görevi zamanlama yapmaktadır.
- İşlemci frekanasına bağlı olarak çalışırlar.
- Dışarıdan gelen pulse (darbe)leri sayarlar.
- İşlemciye tanıtılan bir süre ile, geçen süreyi karşılaştırma ve belli bir süre sonunda kesme üretme gibi işlemlerde kullanılırlar.
- Çoğu mikrodenetleyicide PWM(Pulse Width Modulation) birimleri de Timer ünitelerine bağlı olarak çalışırlar.
- STM32F4 DISCOVERY işlemcisinde toplam 17 adet timer bulunur.
- 10 adet genel amaçlı timer
- 2 adet gelişmiş timer
- 1 adet IWDG (Independent Watchdog) timer
- 1 adet WWDG (Window Watchdog) timer
- 1 adet Systemtick (Sistem Zamanlayıcısı) timer var.

Timer	Özellik
TIM1ve,TIM8	16 bit gelişmiş timer
TIM3, TIM4, TIM9, TIM10, TIM11, TIM12, TIM13, TIM14	16 bit genel amaçlı timer
TIM2, TIM5	32 bit genel amaçlı timer
TIM6, TIM7	16 bit basit timer

1- Genel Amaçlı Timer2, Timer3, Timer4 Ve Timer5 Birimleri

- Timer 2,3,4 ve timer5 birimleri, düşük hızlı APB1 (42 MHz) veri yolu üzerinde bulunmaktadır. Eğer APB1 prescaler (bölgücü) değeri 1 den farklı ise bu timerların clock frekansları beslendikleri frekansların 2 katına çıkar. Yani 84 MHz clock frekansına sahip olur.
- Timer3(TIM3) ve Timer4(TIM4) 16-bit lik sayıcıya, Timer2(TIM2) ve Timer5(TIM5) 32-bit lik sayıcıya sahiptirler.
- Bu sayıcılar (up)yukarı, (down)aşağı ve (auto-reload)hizalanmış/merkezlenmiş modlarda sayma yapabilirler.
- Ayrıca bu sayıcıların otomatik yükleme özellikleri de vardır.
- 16-bit genişliğinde kontrol edilebilir ön bölgücü değeri (prescaler) vardır.
- Bu timer biriminde 4x16 adet yüksek çözünürlüktü capture/compare kanalı bulunur. Bu kanallar;
 - > Giriş Yakalama (Input Capture)
 - > Çıkış Karşılaştırma (Output Compare)
 - > Darbe Genişlik Modülasyonu (PWM)
 - > Tek Darbe Çıkışı (One-Pulse)
- Dahili diğer Timer birimleri ile sonkronizasyon
- Interrupt (kesme) ve DMA üretimi
- Clock kaynağı seçimi

2- Basic(Basit) Timer Timer6 ve Timer7

- Timer 6 ve Timer 7 birimleri birbirinden tamamen bağımsızdır. Ortak register veya ortak veri kullanımı gibi bir durum yoktur.
- Basic Timer birimleri genel sayaç olarak kullanılabilecekleri gibi, spesifil olarak Dijital Analog Çevirici (DAC) biriminin tetikleyicisi olarak da kullanılabilir.
- 16-bit genişliğinde otomatik geri yüklenen artan sayaca sahiptir. (auto-reload upcounter)
- 16-bit genişliğinde kontrol edilebilir ön bölgücü(Prescaler) değere sahiptir.
- DAC birimi için tetikleme çıkışlarına sahiptir.
- Interrupt ve DMA üretimi mevcuttur.
- Çalışma prensibi genel amaçlı timer'ların çalışma prensibi ile aynıdır.

3- Gelişmiş Timer 1 Ve Timer 8 Birimleri

- Timer 1 ve Timer 8, yüksek hızlı APB2 veri yolu (84 MHz) üzerinde bulunmaktadır. Eğer APB2 prescaler değişkeni "1" değerinden farklı ise bu timer birimlerinin saat frekansı, APB2'nin frekans değerinin iki katı olur. Yani, bu timer birimlerinin maksimum çalışma frekansları 168 MHz olabilir.
- Timer 1 ve Timer 8 birimleri 16 bitlik sayıcıya sahiptirler.
- Bu sayıcılar; yukarı, aşağı ve merkezlenmiş modlarda sayma yapabilirler.
- Bu sayıcıların otomatik geri yükleme özellikleri bulunmaktadır.
- Bu timer birimlerinde 4x16 adet yüksek çözünürlüklü capture/compare kanalı da bulunur.
 - > Bu kanallar giriş öklär olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.

4- Genel Amaçlı 2 Kanallı Timer 9 ve Timer 12 Birimleri

- Timer 9 yüksek hızlı (84 MHz) APB2 ve Timer 12 düşük hızlı (42 MHz) APB1 üzerinde bulunmaktadır.
- Bu birimlerin frekansları diğerlerinde olduğu gibi veriyolu hızlarının iki katında çalışabilirler.
- Timer 9 ve Timer 12 birimleri 16 bitlik sayıcıya sahiptirler. Bu sayıcılar sadece yukarı sayma yapabilirler. Ayrıca bu sayıcıların otomatik geri yükleme özellikleri de bulunmaktadır.
- Bu timer birimlerinde 2x16 adet yüksek çözünürlüklü capture/compare kanalı da bulunur.
 - > Bu kanallar giriş öķiš olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.

5- Genel Amaçlı 1 Kanallı Timer 10-11 ve Timer 13-14 Birimleri

- Timer 10 ve Timer 11 yüksek hızlı (84 MHz) APB2 ve Timer 13 ve Timer 14 düşük hızlı (42 MHz) APB1 üzerinde bulunmaktadır. Bu birimlerin frekansları diğerlerinde olduğu gibi veriyolu hızlarının iki katında çalışabilirler.
- Bu birimler 16 bitlik sayıcıya sahiptirler. Bu sayıcılar sadece yukarı sayma yapabilirler. Ayrıca bu sayıcıların otomatik geri yükleme özellikleri de bulunmaktadır.
- Bu timer birimlerinde 2x16 adet yüksek çözünürlüklü capture/compare kanalı da bulunur.
 - > Bu kanallar giriş öķiš olarak ayarlanabilir, çıkış karşılaştırabilir, PWM sinyali üretebilir, sinyal yakalayabilir ve harici bir PWM sinyalini algılayabilirler.

6- INDEPENDENT WATCHDOG(IWDG) BİRİMİ

- IWDG, işlemci saatinden bağımsız, kendine ait dahili RC osilatörden (LSI 32 KHz) beslenen bir watchdog timeridir.
- Watchdog kelimesinin Türkçe karşılığı, bekçi köpeği demektir.
- Mikrodenetleyici içerisindeki amacı da bekçilik yapmaktadır.
 - > Peki neye bekçilik yapacak?
- Mikrodenetleyici, harici sebeplerden veya kodlardaki bir hata sebebiyle kilitlenebilir. Mikrodenetleyici kilitlendiğinde, yürütüğü işlemler durur. Bu tür durumlarda mikrodenetleyicinin tekrar başlatılması gereklidir. İşte watchdog timerlar burada devreye girerler. Watchdog timerlarda belirlenen bir süre sonunda sıfırlanırlar ve işlemciyi resetlerler.

7- WINDOW WATCHDOG(WWDG) BİRİMİ

- WWDG(Pencere Watchdog) birimi belirli bir pencere içerisinde counter (sayıcı) kaydedicisine tekrar değer yüklenebildiği için bu isimle anılmaktadır.
- Ayarlanabilir süre penceresine sahiptir.
- Anormal erken ve anormal geç uygulama davranışını algılayabilir.
- Önceden belirlenen duruma göre işlemciyi resetler.

07_01 Timer Değer Okuma

25 Aralık 2021 Cumartesi 00:59

07_01 Timer Değer Okuma

➤ SPL

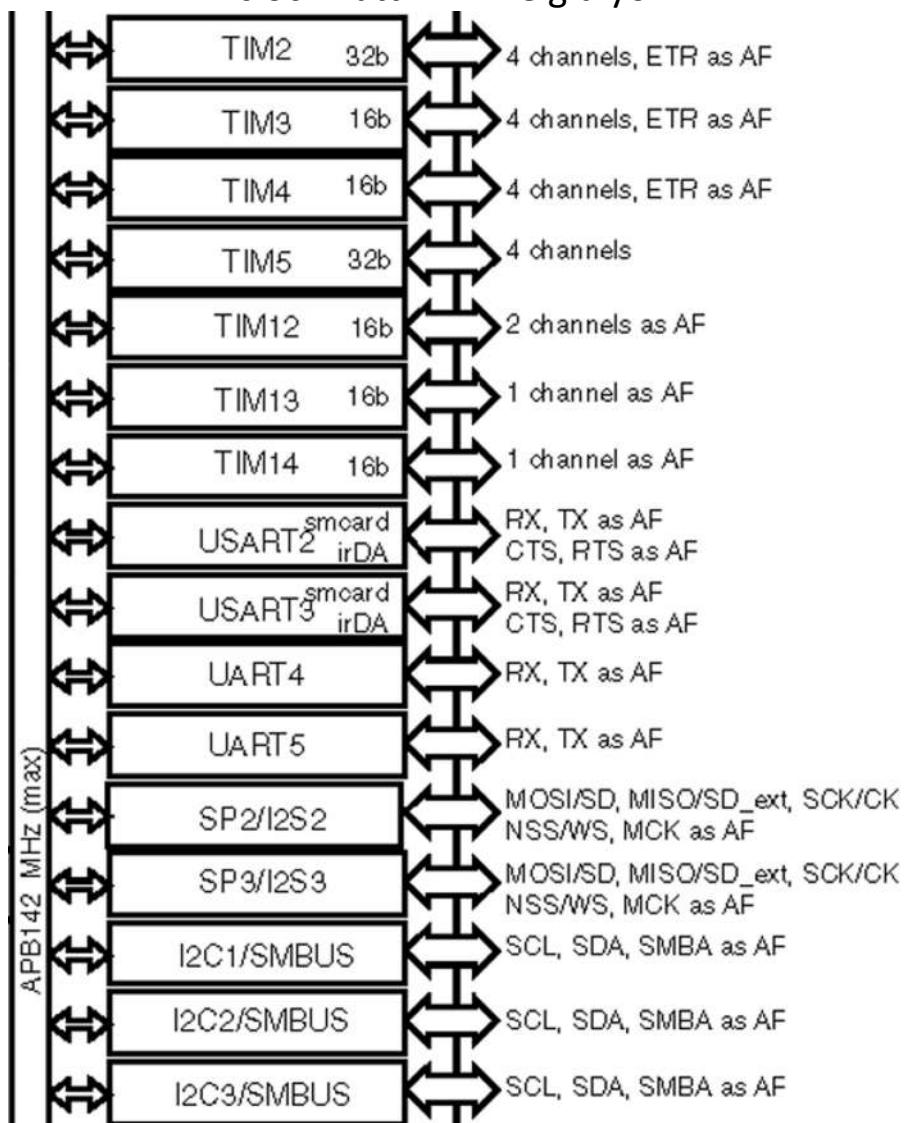
Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

- TIM2'nin clock hattı APB1'e gidiyor.



RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved		DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
		rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw		rw
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN			
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

0.biti 1 yapıyoruz.

Bit 0 **TIM2EN**: TIM2 clock enable

Set and cleared by software.

0: TIM2 clock disabled

1: TIM2 clock enabled

`RCC->APB1ENR |= 0x00000001;`

TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							CKD[1:0]	ARPE	CMS	DIR	OPM	URS	UDIS	CEN		
							rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Sayma yapacağımdan saymayı başlatmak için 0.biti aktif ediyoruz. Sayma başlayacağından bu işlem fonksiyonun en son satırında olmalı.

Bit 0 **CEN**: Counter enable

0: Counter disabled

1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

`TIM2->CR1 |= 1 << 0;`

- Saymanın yönünü belirliyoruz. Biz yukarı doğru saymasını istiyoruz.

Bit 4 **DIR**: Direction

0: Counter used as upcounter

1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

`TIM2->CR1 |= 0 << 4;`

- 5. ve 6.biti 0 yapıyoruz.

Bits 6:5 **CMS**: Center-aligned mode selection

00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).

01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.

10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.

11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

- Sistem clock 168MHz ayarlamıştık ve TIM2 clock veri yolu ise 42MHz'dir fakat bunun 2 katı değeri alıyorlardı yani 84MHz'dir. Biz bunu bölmek istiyorsak ayarlayabiliyoruz. Biz bölmüyoruz.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

00: $t_{DTS} = t_{CK_INT}$

01: $t_{DTS} = 2 \times t_{CK_INT}$

10: $t_{DTS} = 4 \times t_{CK_INT}$

11: Reserved

- 8. ve 9.biti 0 yapıyoruz.

TIM2->**CR1** |= 0 << 8;

TIMx slave mode control register (TIMx_SMCR)

Address offset: 0x08

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ETP	ECE	ETPS[1:0]		ETF[3:0]				MSM	TS[2:0]			Res.	SMS[2:0]		
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw	rw

- Slave mode çalışmayacağız.

Bits 2:0 **SMS**: Slave mode selection

When external signals are selected the active edge of the trigger signal (TRGI) is linked to the polarity selected on the external input (see Input Control register and Control Register description).

000: Slave mode disabled - if CEN = '1 then the prescaler is clocked directly by the internal clock.

001: Encoder mode 1 - Counter counts up/down on TI2FP1 edge depending on TI1FP2 level.

010: Encoder mode 2 - Counter counts up/down on TI1FP2 edge depending on TI2FP1 level.

011: Encoder mode 3 - Counter counts up/down on both TI1FP1 and TI2FP2 edges depending on the level of the other input.

100: Reset Mode - Rising edge of the selected trigger input (TRGI) reinitializes the counter and generates an update of the registers.

101: Gated Mode - The counter clock is enabled when the trigger input (TRGI) is high. The counter stops (but is not reset) as soon as the trigger becomes low. Both start and stop of the counter are controlled.

110: Trigger Mode - The counter starts at a rising edge of the trigger TRGI (but it is not reset). Only the start of the counter is controlled.

111: External Clock Mode 1 - Rising edges of the selected trigger (TRGI) clock the counter.

Note: The gated mode must not be used if TI1F_ED is selected as the trigger input (TS=100). Indeed, TI1F_ED outputs 1 pulse for each transition on TI1F, whereas the gated mode checks the level of the trigger signal.

The clock of the slave timer must be enabled prior to receiving events from the master timer, and must not be changed on-the-fly while triggers are received from the master timer.

`TIM2->SMCR |= 0 << 0;`

TIMx event generation register (TIMx_EGR)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
Reserved										TG	Res.	CC4G	CC3G	CC2G	CC1G	UG
w									w	w	w	w	w	w	w	

- Sayma dolduğunda sıfırlaması için 1 yaparız. 0.bit 1 yapıyoruz.

Bit 0 **UG**: Update generation

This bit can be set by software, it is automatically cleared by hardware.

0: No action

1: Re-initialize the counter and generates an update of the registers. Note that the prescaler counter is cleared too (anyway the prescaler ratio is not affected). The counter is cleared if the center-aligned mode is selected or if DIR=0 (upcounting), else it takes the auto-reload value (TIMx_ARR) if DIR=1 (downcounting).

`TIM2->EGR |= 1 << 0;`

TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Prescaler bizim sayısının en üst seviyesini belirler.
- Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.
- TIM2 clock hızı 84MHz olduğundan bunu kaç'a bölmeliyim diye soruyoruz. Bu clock hız için 42000'e bölüyoruz. Bu sayı da 41999 sayısı yapıyor.

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in "reset mode").

TIM2->**PSC** |= 41999;

- 84MHz'i 42000'e böldüğümüzde 2000 sayısı yapıyor. Bu değer auto-reload oluyor. Bunun 1 eksigini yazıyoruz.

TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ARR[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ARR[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2 and TIM5).

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

TIM2->**ARR** |= 1999;

- Böylece 1 sn'de 2000'e kadar sayıyor.

- RCC ve TIM için yazdığımız fonksiyonlar aşağıdaki gibidir.

```
5 void RCC_Config(void)
6 {
7     RCC->CR |= 0x00010000; //HSEON
8     while(!(RCC->CR & 0x00020000)); //HSERDY
9     RCC->CR |= 0x00080000; //CSSON
10    RCC->CFGR = 0x00000000;
11    RCC->PLLCFGR |= 0x00400000; //PLLSRC
12    RCC->PLLCFGR |= 0x00000004; //PLLM 4
13    RCC->PLLCFGR |= 0x00002A00; //PLLN 168
14    RCC->PLLCFGR |= 0x00000000; //PLLP 2
15    RCC->CR |= 0x01000000; //PLLON
16    while(!(RCC->CR & 0x02000000)); //PLLRDY
17    RCC->CFGR |= 0x00000001; //SW
18    while(!(RCC->CR & 0x00000001)); //SWS
19    RCC->CFGR |= 0x00000000; //HPRE AHB 1
20    RCC->CFGR |= 0x00001400; //PPRE1 APB1 4
21    RCC->CFGR |= 0x00008000; //PPRE2 APB2 2
22    RCC->CIR |= 0x00080000; //HSERDYC
23    RCC->CIR |= 0x00800000; //CSSC
24 }
```

```

26 void TIM_Config(void)
27 {
28     RCC->APB1ENR |= 0x00000001;           //TIM2EN
29
30     TIM2->CR1 |= 0 << 4;                //DIR Counter used as up counter
31     TIM2->CR1 |= 0 << 5;                //CMS Edge-aligned mode
32     TIM2->CR1 |= 0 << 8;                //tDTS = tCK_INT 84MHz
33     TIM2->SMCR |= 0 << 0;              //SMS Slave mode disabled
34     TIM2->EGR |= 1 << 0;                //UG Update generation
35     TIM2->PSC |= 41999;                 //PSC Prescaler value
36     TIM2->ARR |= 1999;                  //ARR Auto-reload value
37     TIM2->CR1 |= 1 << 0;                //CEN Counter enabled
38 }

```

Kod Kısmı

TIMx counter (TIMx_CNT)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CNT[31:16] (depending on timers)															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CNT[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- CNT registeri sayma işlemini burada tutuyor.
- Değişken atayıp bu değişkene CNT registerine eşitliyoruz.

```
3 uint16_t count = 0;
```

```

40 int main(void)
41 {
42     RCC_Config();
43     TIM_Config();
44
45     while (1)
46     {
47         count = TIM2->CNT;
48     }
49 }
50 }

```

Variable Name	Address/Expression	Read Value
count	0x2000002c	1838

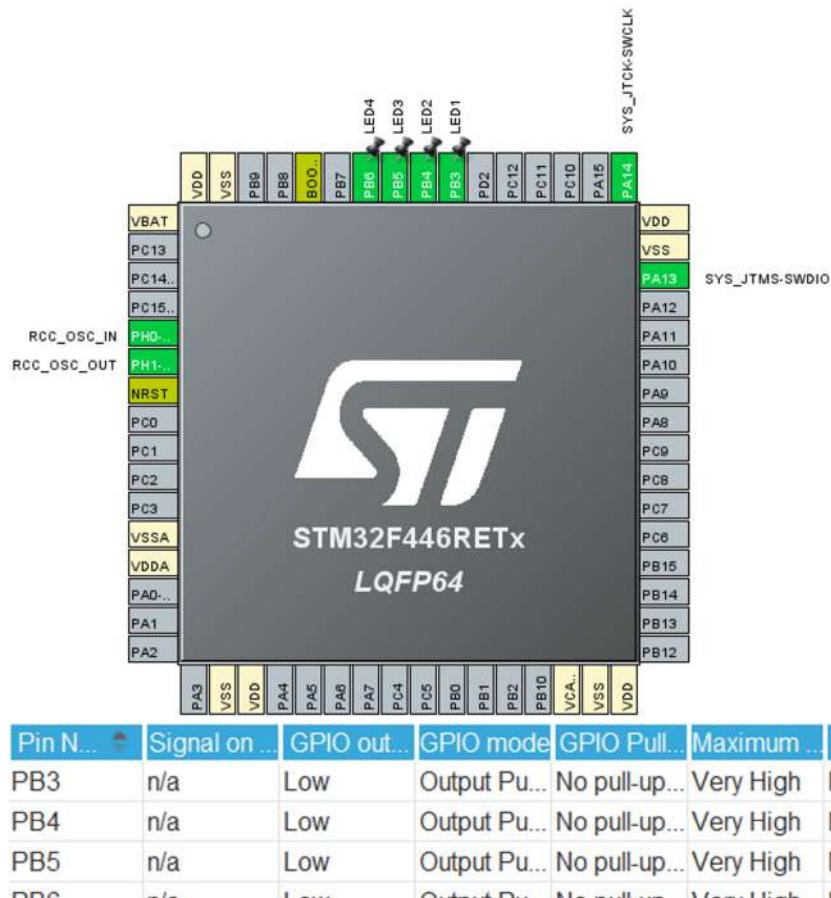
07_02 Timer Interrupt

25 Aralık 2021 Cumartesi 00:59

07_02 Timer Interrupt

➤ HAL

Konfigürasyon Kısmı



- Timers kısmından TIM2 seçimi yapılır. Ardından Mod kısmından sadece Clock Source kısmını Internal Clock yaparız.

Slave Mode

Trigger Source

Clock Source

Channel1

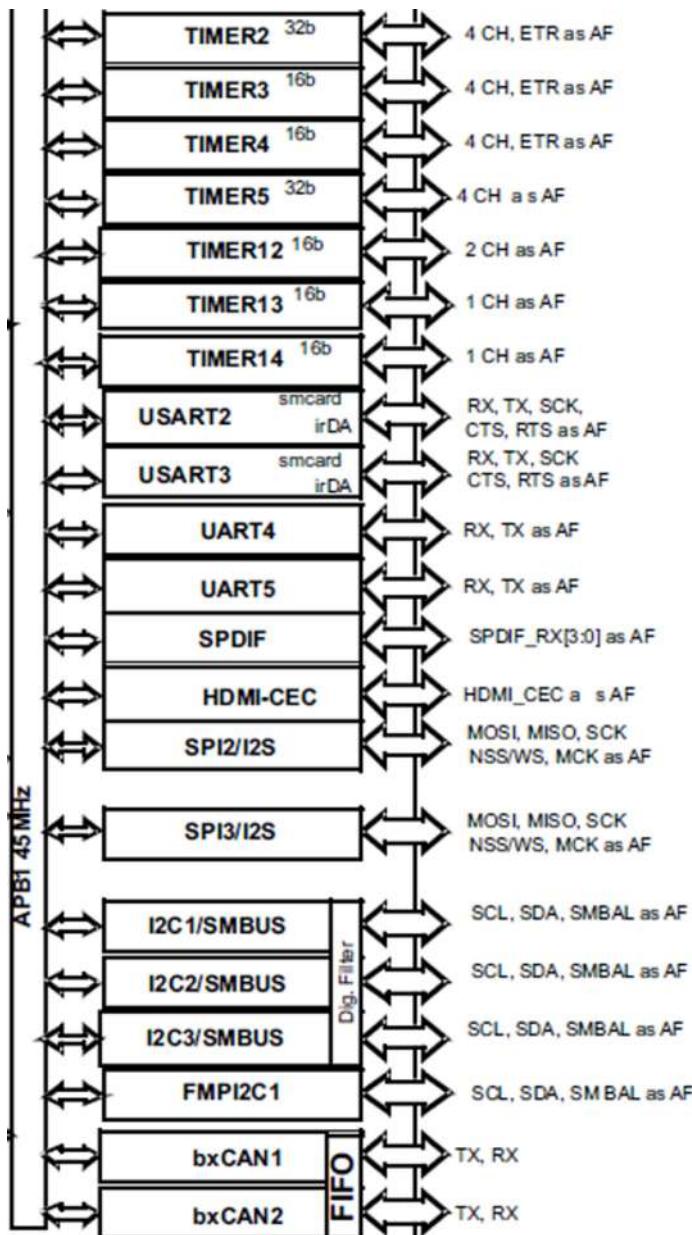
Channel2

Channel3

Channel4

Combined Channels

- Sistem clock 180MHz ayarlamıştık ve TIM2 clock veri yolu ise 45MHz'dir. fakat bunun 2 katı değeri alıyorlardı yani 90MHz'dir.



- Daha sonra Parameter Settings'den TIM2 1 saniye aralıklarla tekrarlı şekilde çalıştıracak değerler girilir. Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.
- Prescalaler bizim sayısının en üst seviyesini belirler. Burası 16 bit olduğundan en fazla 65535 yazabilirim. TIM2 clock hızı 90MHz olduğundan bunu kaça bölmeliyim diye soruyoruz. Bu clock hızı için 45000'e bölüyoruz. Bu sayı da 44999 sayısı yapıyor.
- 90MHz'i 45000'e böldüğümüzde 2000 sayısı yapıyor. Bu değer Auto-reload oluyor. Bunun da 1 eksigini yazıyoruz. Counter Period kısmında her seferinde taşıma işlemi bittiğinden sonra tekrar bunu yükler. Yükleyeceği değeri yazarız.
- Sayma şeklini Up belirleyip yukarı doğru sayıyor.
- Auto-reload preload kısmında Enabled diyerek sayma bittiğinde başa dönmesini sağlarız.

- 2000 değeri yazmasaydık. Sonuç 2000 Hz olacağından sonucunda 0,0005s yani 0,5 ms olacaktır.
- 1 sn=1000 ms

Counter Settings

Prescaler (PSC - 16 bits value)	45000-1
Counter Mode	Up
Counter Period (AutoReload Register -	.2000-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Enable

$$UpdateEvent = \frac{Timer_{clock}}{(Prescaler + 1)(Period + 1)}$$

$$UpdateEvent = \frac{90.000.000}{(45000)(2000)} = 1 \text{ Hz} = \frac{1}{1} \text{ s} = 1 \text{ s}$$

- NVIC Settings kısmından TIM2 global interrupt Enabled yapılır. Bununla her güncellemede, sayıyı bitirmede bir interrupt oluşmasını sağlıyoruz.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
TIM2 global interrupt	<input checked="" type="checkbox"/>	0	0

Kod Kısımları

- hal_tim.c dosyasından Timer'ı Interrupt ile başlatmamız gerekiyor.

```

453 /**
454  * @brief Starts the TIM Base generation in interrupt mode.
455  * @param htim TIM Base handle
456  * @retval HAL status
457 */
458 HAL_StatusTypeDef HAL_TIM_Base_Start_IT(TIM_HandleTypeDef *htim)

```

```

95  /* USER CODE BEGIN 2 */
96  HAL_TIM_Base_Start_IT(&htim2);
97  /* USER CODE END 2 */

```

- 96.satır çalışınca it.c dosyasına gider. Burada TIM için geçerli fonksiyonu 1sn'de bir çalıştırır.

```

202 /**
203  * @brief This function handles TIM2 global interrupt.
204 */
205 void TIM2_IRQHandler(void)
206 {
207  /* USER CODE BEGIN TIM2_IRQHandler_0 */
208
209  /* USER CODE END TIM2_IRQHandler_0 */
210  HAL_TIM_IRQHandler(&htim2);
211  /* USER CODE BEGIN TIM2_IRQHandler_1 */
212
213  /* USER CODE END TIM2_IRQHandler_1 */
214 }

```

- 210.satır ile bu fonksiyon dallanır ve dallanan fonksiyon içerisinde CallBack fonksiyonu vardır. Bu fonksiyon sayesinde int main içerisinde de kodlarımlı yazabilirim.

```
2038/** @defgroup TIM_Exported_Functions_Group9 TIM Callbacks functions
2039 * @brief   TIM Callbacks functions
2040 * @{
2041 */
2042 /* Callback in non blocking modes (Interrupt and DMA) *****/
2043 void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim);

57/* Private user code -----
58 /* USER CODE BEGIN 0 */
59void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
60 {
61     HAL_GPIO_TogglePin(GPIOB, LED1_Pin | LED2_Pin | LED3_Pin | LED4_Pin);
62 }
63 /* USER CODE END 0 */
• 1sn aralıklarla ledi yakıp söndürüyor.
```

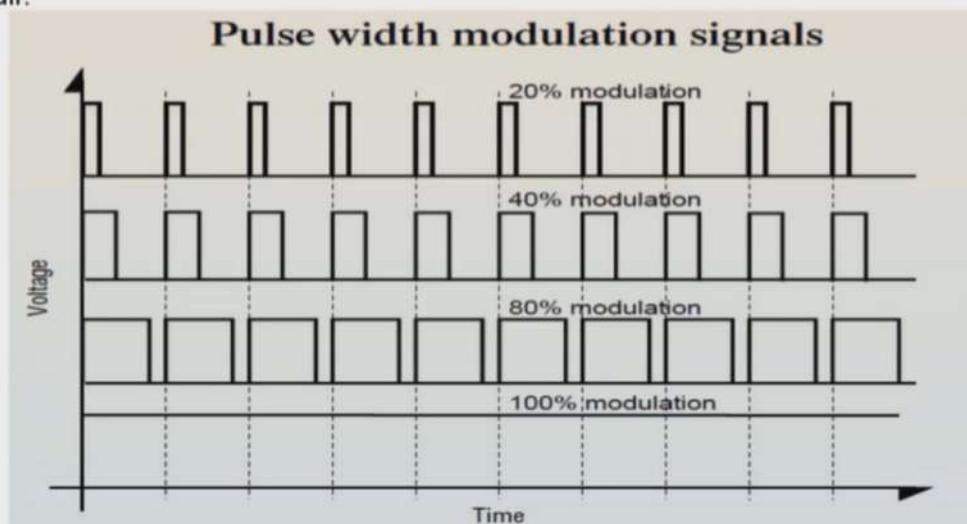
08 PWM

25 Haziran 2021 Cuma 23:48

08 PWM

Giriş

- Pulse Width Modulation (Darbe genişlik modülasyonu); bir kare dalga sinyalinin, yüksek seviyede kalma süresine müdahale ederek, bu sinyalin geriliminin ortalama değerinin değiştirilmesi olarak tanımlanabilir.
- PWM endüstride iletişim, motor kontrol, ısıtma, aydınlatma gibi önemli bir çok alanda kullanılmaktadır.
- Aşağıdaki şikilde sinyalin görev yapan kısmın (duty cycle) süresinin değiştirilmesi, yani PWM olayını göstermektedir.



1. **Mod 1:** Yukarı doğru sayarken CNT < CCRx (Capture Compare Register) dan düşükse kanal aktif, diğer durumda pasif olur. Aşağı doğru sayarken CNT > CCRx ise kanal pasif, değilse aktif olur.
 2. **Mod 2:** Yukarı doğru sayarken CNT < CCRx (Capture Compare Register) dan düşükse kanal pasif, diğer durumda aktif olur. Aşağı doğru sayarken CNT > CCRx is kanal aktif, değilse pasif olur.
- PWM frekansını hesaplamak için, aşağıdaki formüllerden yararlanmamız lazım;
 - > $\text{Period} = (\text{Timer_Tick_Freq} / \text{PWM_Freq}) - 1$
 - > $\text{PWM_Freq} = \text{Timer_Tick_Freq} / (\text{Period} + 1)$
 - > $\text{Timer_Tick_Freq} = \text{Timer_CLK} / (\text{Prescaler} + 1)$
 - Buradan şunu düşünmeliyiz. Timer frekansı kullanıcı tarafından belirlenir. Aynı zamanda PWM de istenilen frekansta çalışılacağı düşünülecek olursa, bizim belirleyeceğimiz iki değer var. Bunlardan biri prescaler, diğeri ise period. Aslında temel olarak PWM in istenilen frekansta çalışması için prescaler değeri küçük bir değer seçilir ve period bu değere göre ayarlanır.

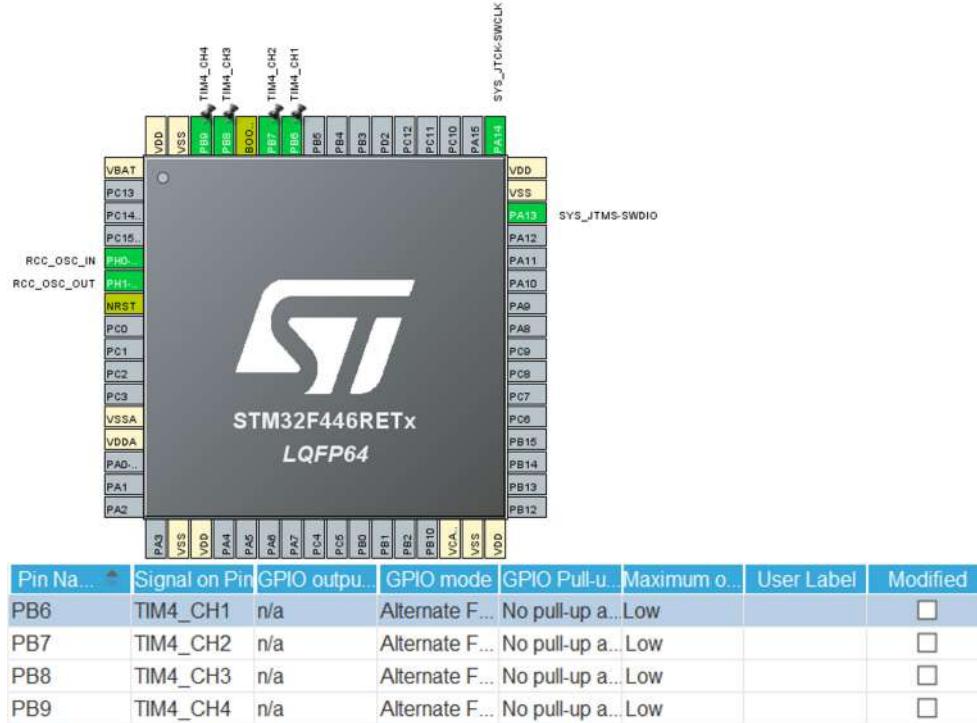
08_01 PWM Kullanımı

25 Aralık 2021 Cumartesi 00:59

08_01 PWM Kullanımı

➤ HAL

Konfigürasyon Kısmı



- Timers kısmından TIM4 seçimi yapılır. Ardından Mod kısmından kanal seçimi yapılır.

Slave Mode: Disable

Trigger Source: Disable

Internal Clock

Channel1: PWM Generation CH1

Channel2: PWM Generation CH2

Channel3: PWM Generation CH3

Channel4: Forced Output CH4

Combined Channels: Disable

XOR activation

- Period değerimize göre kanal çıkışlarına Pulse değeri yazacağız.
- Period kısmını Duty Cycle en fazla 100 olduğundan Period kısmına 100-1 olarak gireriz.

Yani Period 100 ve Pulse değeri 50 girersek aslında %50 Duty Cycle olur.

- 10kHz için işlem sonucunda Prescaler 90000 girilir.
- 1 kHz=1000Hz

$$UpdateEvent = \frac{90.000.000}{(Prescaler + 1)(100)} = 10000 Hz = 10 kHz$$

Prescaler + 1 = 90

Counter Settings

Prescaler (PSC - 16 bits value)	90-1
Counter Mode	Up
Counter Period (AutoReload Register - 16 b)	100-1
Internal Clock Division (CKD)	No Division
auto-reload preload	Disable

PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 2	
Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 3	
Mode	PWM mode 1
Pulse (16 bits value)	0
Output compare preload	Enable
Fast Mode	Disable
CH Polarity	High
Forced Output Channel 4	
Mode	Forced Active
Pulse (16 bits value)	0
Output compare preload	Disable
CH Polarity	High

- Mode 1 ile Mode 2 arasındaki fark High ile Low durumların tersi olmasına.
- Kanal %25 ise Mode durumunda %25'te High %75'te Low oluyordu.
%75 olduğunda %25'te Low %75'te High oluyor.
- Fast Mode durumun Disable olduğu zaman saymaya yaparken üst limite kadar sayıktan sonra 0'a doğru azala azala inerken Enabled olduğu zaman 0'a doğru birden iner.

Kod Kısmı

- hal_tim.c dosyasından Timer'ı PWM ile başlatmamız gerekiyor.

```

1439 /**
1440  * @brief Starts the PWM signal generation.
1441  * @param htim TIM handle
1442  * @param Channel TIM Channels to be enabled
1443  *   This parameter can be one of the following values:
1444  *     @arg TIM_CHANNEL_1: TIM Channel 1 selected
1445  *     @arg TIM_CHANNEL_2: TIM Channel 2 selected
1446  *     @arg TIM_CHANNEL_3: TIM Channel 3 selected
1447  *     @arg TIM_CHANNEL_4: TIM Channel 4 selected
1448  * @retval HAL status
1449 */
150 HAL_StatusTypeDef HAL_TIM_PWM_Start(TIM_HandleTypeDef *htim, uint32_t Channel)

91 /* USER CODE BEGIN 2 */
92 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_1);
93 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_2);
94 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_3);
95 HAL_TIM_PWM_Start(&htim4, TIM_CHANNEL_4);
96 /* USER CODE END 2 */

45 /* USER CODE BEGIN PV */
46 int i;
47 /* USER CODE END PV */

```

```

1375 /**
1376  * @brief Set the TIM Capture Compare Register value on runtime without calling another time ConfigChannel function.
1377  * @param __HANDLE__ TIM handle.
1378  * @param __CHANNEL__ TIM Channels to be configured.
1379  *   This parameter can be one of the following values:
1380  *     @arg TIM_CHANNEL_1: TIM Channel 1 selected
1381  *     @arg TIM_CHANNEL_2: TIM Channel 2 selected
1382  *     @arg TIM_CHANNEL_3: TIM Channel 3 selected
1383  *     @arg TIM_CHANNEL_4: TIM Channel 4 selected
1384  * @param __COMPARE__ specifies the Capture Compare register new value.
1385  * @retval None
1386 */
1387 #define __HAL_TIM_SET_COMPARE(__HANDLE__, __CHANNEL__, __COMPARE__) \
1388 (((__CHANNEL__) == TIM_CHANNEL_1) ? ((__HANDLE__)->Instance->CCR1 = (__COMPARE__)) :\
1389 ((__CHANNEL__) == TIM_CHANNEL_2) ? ((__HANDLE__)->Instance->CCR2 = (__COMPARE__)) :\
1390 ((__CHANNEL__) == TIM_CHANNEL_3) ? ((__HANDLE__)->Instance->CCR3 = (__COMPARE__)) :\
1391 ((__HANDLE__)->Instance->CCR4 = (__COMPARE__)))

```

- Compare kısmına Pulse değerini yazıyoruz.

```

98 /* Infinite loop */
99 /* USER CODE BEGIN WHILE */
100 while (1)
101 {
102     /* USER CODE END WHILE */
103
104     /* USER CODE BEGIN 3 */
105     for(i=0;i<=1999;i++)
106     {
107         __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_1,i);
108         __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_2,i);
109         __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_3,i);
110         __HAL_TIM_SET_COMPARE(&htim4,TIM_CHANNEL_4,i);
111         HAL_Delay(100);
112     }
113
114 }
115 /* USER CODE END 3 */
116 }

```

➤ SPL

Konfigürasyon Kısımları

Kod Kısımları

➤ REGISTER

Konfigürasyon Kısımları

- RCC için yazdığımız fonksiyonlar aşağıdaki gibidir.

```

3 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;    //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;    //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;    //PLLP 2
13    RCC->CR |= 0x01000000;          //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;        //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;        //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;        //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;        //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;         //HSERDYC
21    RCC->CIR |= 0x00800000;         //CSSC
22 }

```

- GPIO için çıkışlarını alternatif fonksiyon yapıyoruz.

GPIO port mode register (GPIOx_MODER) ($x = A..I/J/K$)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Kart üzerindeki ledleri kullanıyoruz. 24., 26., 28. ve 30.biti 1 yapıyoruz.

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits ($y = 0..15$)

These bits are written by software to configure the I/O direction mode.
 00: Input (reset state)
 01: General purpose output mode
 10: Alternate function mode
 11: Analog mode

GPIOD->MODER |= 2 << 24 | 2 << 26 | 2 << 28 | 2 << 30;

- Alternatif fonksiyon ile kast edilen pinin çevresel birimlerden I2C, SPI olarak kullanılacağını belirtiyor. Biz burada TIM4 için kullanacağımızı belirteceğiz.

GPIO alternate function low register (GPIOx_AFRL) ($x = A..I/J/K$)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO alternate function high register (GPIOx_AFRH) ($x = A..I/J$)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Low ile 0 ile 7.pinler arası iken high 8 ile 15.pinler arasıdır. Biz 12, 13, 14 ve 15. pinleri kullandığımızdan high olanı kullanıyoruz.

Bits 31:0 **AFRHy**: Alternate function selection for port x bit y ($y = 8..15$)

These bits are written by software to configure alternate function I/Os

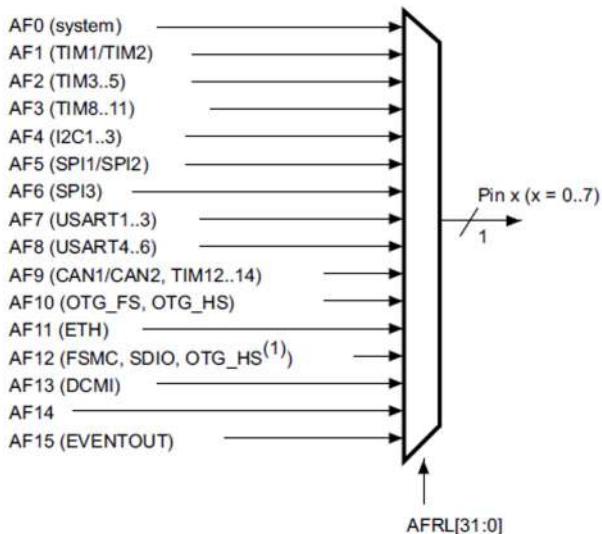
AFRH selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

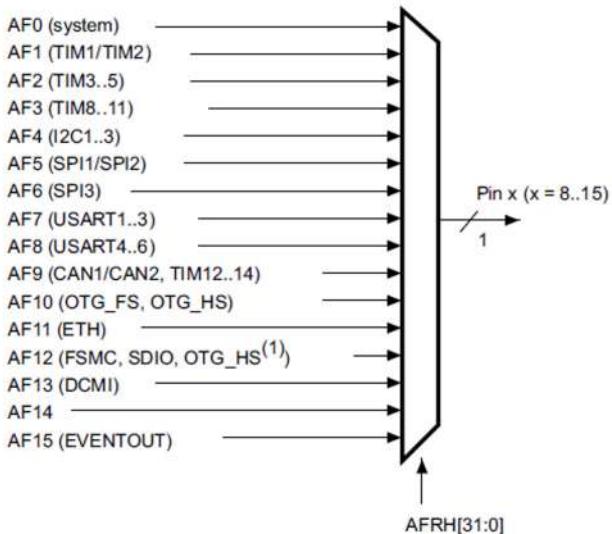
- Seçtiğimiz TIM4 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

Selecting an alternate function

For pins 0 to 7, the GPIOx_AFRL[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx_AFRH[31:0] register selects the dedicated alternate function



ai17538

- TIM4, AF2'de bulunuyor. Böylece pinlere AF2 için olan 0010 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunda belirtmemiz gerekiyor. AFR'ye Ctrl ile sağ tıklarız. AFR'nin parantez içinde 2 elemanlı dizi olduğunu gösterir. 0.eleman low, 1.eleman high temsil eder. Biz 1 yazıyoruz.

682 `_IO uint32_t AFR[2]; /*!< GPIO alternate function registers, Address offset: 0x20-0x24 */`

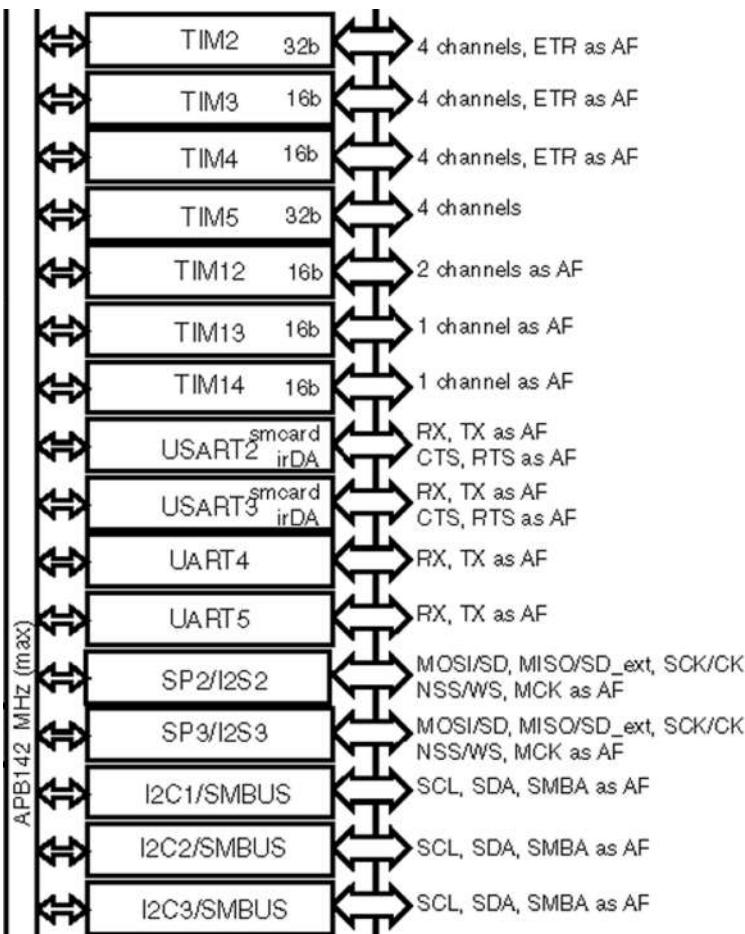
`GPIOD->AFR[1] |= 2 << 16 | 2 << 20 | 2 << 24 | 2 << 28;`

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```

24 void GPIO_Config(void)
25 {
26     RCC->AHB1ENR |= 0x00000008;                                //D clock enable
27
28     GPIOD->MODER |= 2 << 24 | 2 << 26 | 2 << 28 | 2 << 30;      //PD12, PD13, PD14, PD15
29     GPIOD->AFR[1] |= 2 << 16 | 2 << 20 | 2 << 24 | 2 << 28;      //TIM4 AFRH12, AFRH13, AFRH14, AFRH15
30     GPIOD->OTYPER |= 0x00000000;                                //Output push-pull
31     GPIOD->OSPEEDR |= 0xFF000000;                                //Very high speed
32     GPIOD->PUPDR |= 0x00000000;                                //No pull-up, pull-down
33 }
```

- TIM4'nin clock hattı APB1'e gidiyor.



RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

- 2.biti 1 yapıyoruz.

Bit 2 **TIM4EN:** TIM4 clock enable

Set and cleared by software.

0: TIM4 clock disabled

1: TIM4 clock enabled

RCC->APB1ENR |= 0x00000004; //TIM4EN

TIMx control register 1 (TIMx_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						CKD[1:0]		ARPE	CMS		DIR	OPM	URS	UDIS	CEN
						rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Sayma yapacağımından saymayı başlatmak için 0.biti aktif ediyoruz. Sayma başlayacağından bu işlem fonksiyonun en son satırında olmalı.

Bit 0 **CEN**: Counter enable

- 0: Counter disabled
- 1: Counter enabled

Note: External clock, gated mode and encoder mode can work only if the CEN bit has been previously set by software. However trigger mode can set the CEN bit automatically by hardware.

CEN is cleared automatically in one-pulse mode, when an update event occurs.

`TIM4->CR1 |= 1 << 0;`

- Saymanın yönünü belirliyoruz. Biz yukarı doğru saymasını istiyoruz.

Bit 4 **DIR**: Direction

- 0: Counter used as upcounter
- 1: Counter used as downcounter

Note: This bit is read only when the timer is configured in Center-aligned mode or Encoder mode.

`TIM4->CR1 |= 0 << 4;`

- 5. ve 6.biti 0 yapıyoruz.

Bits 6:5 **CMS**: Center-aligned mode selection

- 00: Edge-aligned mode. The counter counts up or down depending on the direction bit (DIR).
- 01: Center-aligned mode 1. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting down.
- 10: Center-aligned mode 2. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set only when the counter is counting up.
- 11: Center-aligned mode 3. The counter counts up and down alternatively. Output compare interrupt flags of channels configured in output (CCxS=00 in TIMx_CCMRx register) are set both when the counter is counting up or down.

Note: It is not allowed to switch from edge-aligned mode to center-aligned mode as long as the counter is enabled (CEN=1)

`TIM4->CR1 |= 0 << 5;`

- Sistem clock 168MHz ayarlamıştık ve TIM4 clock veri yolu ise 42MHz'dir fakat bunun 2 katı değeri alıyorlardı yani 84MHz'dir. Biz bunu bölmek istiyorsak ayarlayabiliyoruz. Biz bölmüyoruz.

Bits 9:8 **CKD**: Clock division

This bit-field indicates the division ratio between the timer clock (CK_INT) frequency and sampling clock used by the digital filters (ETR, TIx),

- 00: $t_{DTS} = t_{CK_INT}$
- 01: $t_{DTS} = 2 \times t_{CK_INT}$
- 10: $t_{DTS} = 4 \times t_{CK_INT}$
- 11: Reserved

- 8. ve 9.biti 0 yapıyoruz.

`TIM4->CR1 |= 0 << 8;`

- PWM aslında capture/compare mode kısmına giriyor.

TIMx capture/compare mode register 1 (TIMx_CCMR1)

Address offset: 0x18

Reset value: 0x0000

The channels can be used in input (capture mode) or in output (compare mode). The direction of a channel is defined by configuring the corresponding CCxS bits. All the other bits of this register have a different function in input and in output mode. For a given bit, OCxx describes its function when the channel is configured in output, ICxx describes its function when the channel is configured in input. Take care that the same bit can have a different meaning for the input stage and for the output stage.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
OC2CE	OC2M[2:0]		OC2PE	OC2FE	CC2S[1:0]	OC1CE	OC1M[2:0]		OC1PE	OC1FE	CC1S[1:0]		IC2F[3:0]	IC2PSC[1:0]	IC1F[3:0]	IC1PSC[1:0]
	IC2F[3:0]						IC1F[3:0]									
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- 1.kanal için output yapıyoruz yani ilk 2 bit 0 yapıyoruz.

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

- 00: CC1 channel is configured as output.
- 01: CC1 channel is configured as input, IC1 is mapped on TI1.
- 10: CC1 channel is configured as input, IC1 is mapped on TI2.
- 11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Bits 1:0 **CC1S**: Capture/Compare 1 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC1 channel is configured as output.

01: CC1 channel is configured as input, IC1 is mapped on TI1.

10: CC1 channel is configured as input, IC1 is mapped on TI2.

11: CC1 channel is configured as input, IC1 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC1S bits are writable only when the channel is OFF (CC1E = 0 in TIMx_CCER).

- Mod olarak PWM mode 1 seçiyoruz. Bunun için bite 110 yazıyoruz.

Bits 6:4 **OC1M**: Output compare 1 mode

These bits define the behavior of the output reference signal OC1REF from which OC1 and OC1N are derived. OC1REF is active high whereas OC1 and OC1N active level depends on CC1P and CC1NP bits.

000: Frozen - The comparison between the output compare register TIMx_CCR1 and the counter TIMx_CNT has no effect on the outputs.(this mode is used to generate a timing base).

001: Set channel 1 to active level on match. OC1REF signal is forced high when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).

010: Set channel 1 to inactive level on match. OC1REF signal is forced low when the counter TIMx_CNT matches the capture/compare register 1 (TIMx_CCR1).

011: Toggle - OC1REF toggles when TIMx_CNT=TIMx_CCR1.

100: Force inactive level - OC1REF is forced low.

101: Force active level - OC1REF is forced high.

110: PWM mode 1 - In upcounting, channel 1 is active as long as TIMx_CNT<TIMx_CCR1 else inactive. In downcounting, channel 1 is inactive (OC1REF='0) as long as TIMx_CNT>TIMx_CCR1 else active (OC1REF=1).

111: PWM mode 2 - In upcounting, channel 1 is inactive as long as TIMx_CNT<TIMx_CCR1 else active. In downcounting, channel 1 is active as long as TIMx_CNT>TIMx_CCR1 else inactive.

Note: In PWM mode 1 or 2, the OCREF level changes only when the result of the comparison changes or when the output compare mode switches from "frozen" mode to "PWM" mode.

- 2.kanal için output yapıyoruz. 8. ve 9.biti 0 yapıyoruz.

Bits 9:8 **CC2S[1:0]**: Capture/Compare 2 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC2 channel is configured as output

01: CC2 channel is configured as input, IC2 is mapped on TI2

10: CC2 channel is configured as input, IC2 is mapped on TI1

11: CC2 channel is configured as input, IC2 is mapped on TRC. This mode is working only if an internal trigger input is selected through the TS bit (TIMx_SMCR register)

Note: CC2S bits are writable only when the channel is OFF (CC2E = 0 in TIMx_CCER).

- 2.kana'lin modunu 1.kanal'da yaptığımız gibi PWM Mode 1 yapıyoruz.

Bits 14:12 **OC2M[2:0]**: Output compare 2 mode

```
TIM4->CCMR1 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;
```

TIMx capture/compare mode register 2 (TIMx_CCMR2)

Address offset: 0x1C

Reset value: 0x0000

Refer to the above CCMR1 register description.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OC4CE	OC4M[2:0]		OC4PE	OC4FE	CC4S[1:0]			OC3CE	OC3M[2:0]			OC3PE	OC3FE	CC3S[1:0]	
	IC4F[3:0]			IC4PSC[1:0]	IC3F[3:0]				IC3PSC[1:0]						
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- CCMR2 ile bu sefer kanal 3 ve 4 için yapıyoruz. Kanal 1 ve 2 ile yaptıklarımızın aynısını yapıyoruz.

Bits 1:0 **CC3S**: Capture/Compare 3 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC3 channel is configured as output

01: CC3 channel is configured as input, IC3 is mapped on TI3

10: CC3 channel is configured as input, IC3 is mapped on TI4

11: CC3 channel is configured as input, IC3 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC3S bits are writable only when the channel is OFF (CC3E = 0 in TIMx_CCER).

Bits 6:4 **OC3M**: Output compare 3 mode

Bits 9:8 **CC4S**: Capture/Compare 4 selection

This bit-field defines the direction of the channel (input/output) as well as the used input.

00: CC4 channel is configured as output

01: CC4 channel is configured as input, IC4 is mapped on TI4

10: CC4 channel is configured as input, IC4 is mapped on TI3

11: CC4 channel is configured as input, IC4 is mapped on TRC. This mode is working only if an internal trigger input is selected through TS bit (TIMx_SMCR register)

Note: CC4S bits are writable only when the channel is OFF (CC4E = 0 in TIMx_CCER).

Bits 14:12 **OC4M**: Output compare 4 mode

TIM4->**CCMR2** |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12;

TIMx capture/compare enable register (TIMx_CCER)

Address offset: 0x20

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CC4NP	Res.	CC4P	CC4E	CC3NP	Res.	CC3P	CC3E	CC2NP	Res.	CC2P	CC2E	CC1NP	Res.	CC1P	CC1E
rw		rw	rw	rw		rw	rw	rw		rw	rw	rw		rw	rw

- Çıkışları Enabled yapıyoruz.

0., 4., 8. ve 12. bitleri 1 yapıyoruz.

Bit 0 **CC1E**: Capture/Compare 1 output enable.

CC1 channel configured as output:

0: Off - OC1 is not active

1: On - OC1 signal is output on the corresponding output pin

CC1 channel configured as input:

This bit determines if a capture of the counter value can actually be done into the input capture/compare register 1 (TIMx_CCR1) or not.

0: Capture disabled

1: Capture enabled

Bit 4 **CC2E**: Capture/Compare 2 output enable.

refer to CC1E description

Bit 8 **CC3E**: Capture/Compare 3 output enable.

refer to CC1E description

Bit 12 **CC4E**: Capture/Compare 4 output enable.

refer to CC1E description

TIM4->**CCER** |= 1 << 0 | 1 << 4 | 1 << 8 | 1 << 12;

TIMx prescaler (TIMx_PSC)

Address offset: 0x28

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSC[15:0]															
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Prescalaler bizim sayısının en üst seviyesini belirler.
- Sayma işlemi 0'dan başlamayıp 1'den başladığından 1 eksigini alarak yazarız.
- TIM4 clock hızı 84MHz olduğundan bunu kaça bölmeliyim diye soruyoruz. Bu clock hız için 42000'e bölüyoruz. Bu sayı da 41999 sayısı yapıyor.

Bits 15:0 **PSC[15:0]**: Prescaler value

The counter clock frequency CK_CNT is equal to $f_{CK_PSC} / (PSC[15:0] + 1)$.

PSC contains the value to be loaded in the active prescaler register at each update event (including when the counter is cleared through UG bit of TIMx_EGR register or through trigger controller when configured in "reset mode").

TIM4->**PSC** |= 41999;

- 84MHz'i 42000'e böldüğümüzde 2000 sayısı yapıyor. Bu değer auto-reload oluyor. Bunun 1 eksigini yazıyoruz.

TIMx auto-reload register (TIMx_ARR)

Address offset: 0x2C

Reset value: 0xFFFF FFFF

Bits 31:16 **ARR[31:16]**: High auto-reload value (on TIM2 and TIM5).

Bits 15:0 **ARR[15:0]**: Auto-reload value

ARR is the value to be loaded in the actual auto-reload register.

Refer to the [Section 18.3.1: Time-base unit](#) for more details about ARR update and behavior.

The counter is blocked while the auto-reload value is null.

TTM4->ARR | = 1999:

- Böylece 1 sn'de 2000'e kadar sayıyor.
 - Pinlere atayacağımız paslar CCR1 12.pin, CCR2 13.pin, CCR3 14.pin ve CCR4 15.pin içindir.

TIMx capture/compare register 1 (TIMx_CCR1)

Address offset: 0x34

Reset value: 0x0000 0000

TIMx capture/compare register 2 (TIMx_CCR2)

Address offset: 0x38

Reset value: 0x0000 0000

TIMx capture/compare register 3 (TIMx_CCR3)

Address offset: 0x3C

Reset value: 0x0000 0000

TIMx capture/compare register 4 (TIMx_CCR4)

Address offset: 0x40

Reset value: 0x0000 0000

- Bunlara atayacağımız değerler en fazla Auto-reload değeri kadar olabilir.

```
TIM4->CCR1 |= 500;
TIM4->CCR2 |= 1000;
TIM4->CCR3 |= 1500;
TIM4->CCR4 |= 1999;
```

- TIM için yazdığımız fonksiyonlar aşağıdaki gibidir.

```
35 void TIM_Config(void)
36 {
37     RCC->APB1ENR |= 0x00000004; //TIM4EN
38
39     TIM4->CR1 |= 0 << 4; //DIR Counter used as up counter
40     TIM4->CR1 |= 0 << 5; //CMS Edge-aligned mode
41     TIM4->CR1 |= 0 << 8; //tDTS = tCK_INT 84MHz
42     TIM4->CCMR1 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12; //Capture/Compare selected output, PWM mode 1 (1 & 2)
43     TIM4->CCMR2 |= 0 << 0 | 6 << 4 | 0 << 8 | 6 << 12; //output, PWM mode 1 selected (3 & 4)
44     TIM4->CCER |= 1 << 0 | 1 << 4 | 1 << 8 | 1 << 12; //output enable (1, 2, 3 & 4 )
45     TIM4->PSC |= 41999; //PSC Prescaler value
46     TIM4->ARR |= 1999; //ARR Auto-reload value
47     TIM4->CR1 |= 500; //PD12
48     TIM4->CCR2 |= 1000; //PD13
49     TIM4->CCR3 |= 1500; //PD14
50     TIM4->CCR4 |= 1999; //PD15
51     TIM4->CR1 |= 1 << 0; //CEN Counter enabled
52 }
```

Kod Kısmı

```
54 int main(void)
55 {
56     RCC_Config();
57     GPIO_Config();
58     TIM_Config();
59
60     while (1)
61     {
62
63     }
64 }
```

- Pindeki ledlerin durumu 12.pin %25, 13.pin %50, 14.pin %75, 15.pin %100 parlaklııkta yanıyor.

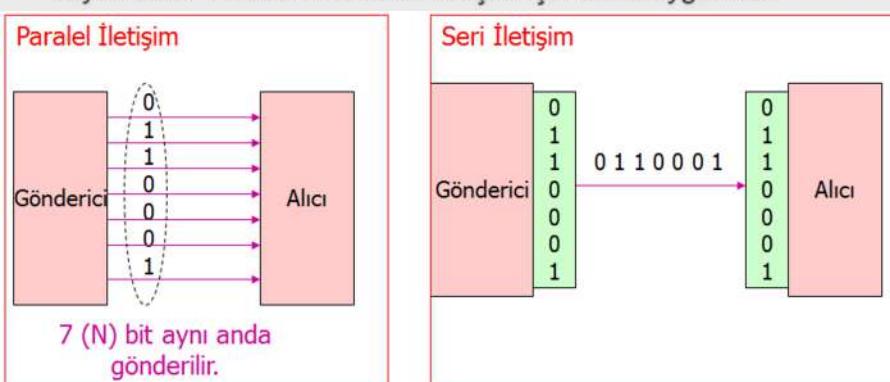
09 UART & USART

5 Mayıs 2021 Çarşamba 08:03

09 UART & USART

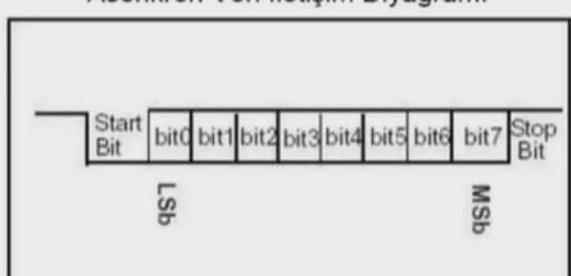
Giriş

- USART (Universal Synchronous - Asynchronous Receiver - Transmitter) 1 ve 0 lardan oluşan verileri, iki dijital sistem arasında alıp-verme işleminde kullanılan bir iletişim protokolüdür.
- Dijital sistemlerde iletişim paralel ve seri olmak üzere iki türlü yapılır. Paralel iletişimde bilgi vericiden alıcıya aynı anda birden fazla bit gidecek şekilde gönderilir. Böylece tek hamlede birden fazla veri karşı tarafa gönderildiği için iletişim hızı yüksektir. Fakat bu iletişim türünde kullanılan hat sayısı fazladır ve uzun mesafeler için uygun değildir.
- Seri iletişimde ise vericiden alıcıya gönderilecek bilgi, tek hat üzerinden sırayla gönderilir. Bu şekilde giden bilginin, tek hamlede tek biti gönderebileceği için iletişim hızı yavaşlatır. Fakat seri iletişimde hat sayısı azdır ve uzun mesafeli iletişim için daha uygundur.



- USART protokolü bir seri iletişim protokolüdür. Usart protokolünde veriler senkron veya asenkron olarak alınabilirler. Senkron veri alışverişinde bir data hattı ve bir clock hattı bulunmalıdır. Daha hattından gidecek veriler, clock hattından gönderilen sinalın her düşen veya yükselen kenarında alıcıya ilettilir.
- Asenkron modda ise verilerin ilettilmesinde bir clock hattına ihtiyaç duyulmaz. Verilerin gönderilmeye başlayacağı, alıcıya bir başlangıç (start) sinyali ile bildirilir ve hemen arkasından veriler akmeye başlar. Burada verilerin gönderim ve alım hızının zamanlanması, alıcının her gelen biti algılayabilmesi açısından çok önemlidir.
- Asenkron seri iletişimde verinin gönderilmeye başlayacağını bildiren bir start biti, verinin gönderilmesinin bittiğini belirten bir stop biti, bir de verilerin doğru olarak gönderilip gönderilmediğini anlamak için kullanılan "parity (eşitlik)" biti bulunmaktadır. Veri bitleri ise 7 veya 8 bit olabilir.
 - > Parity bitinin gönderilmesi şart değildir.
- Asenkron iletişimde kullanılan diğer bir kavram ise veri gönderim hızıdır. Bu kavram "bps" (bits per second - saniyede gönderilen bit sayısı) birimi ile anılır. Standart veri gönderme hızları 110, 150, 300, 600, 1200, 2400, 4800, 9600, 56000, 115200 gibi hızdadır.
- Start biti "0" stop biti "1" verilerinden oluşmaktadır.

Asenkron Veri İletişim Diyagramı



09_01 USART ile Mesaj Gönderme

25 Aralık 2021 Cumartesi 01:00

09_01 USART ile Mesaj Gönderme

➤ HAL

Konfigürasyon Kısımları

Kod Kısımları

➤ SPL

Konfigürasyon Kısımları

Kod Kısımları

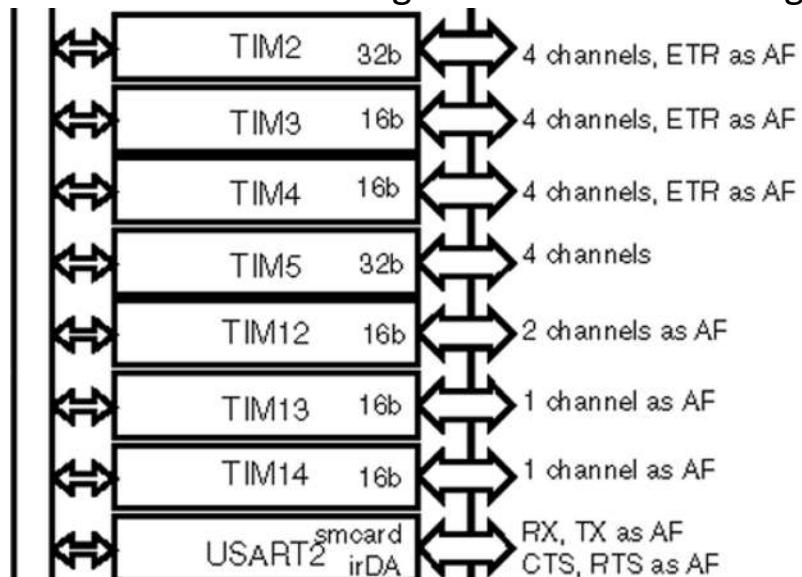
➤ REGISTER

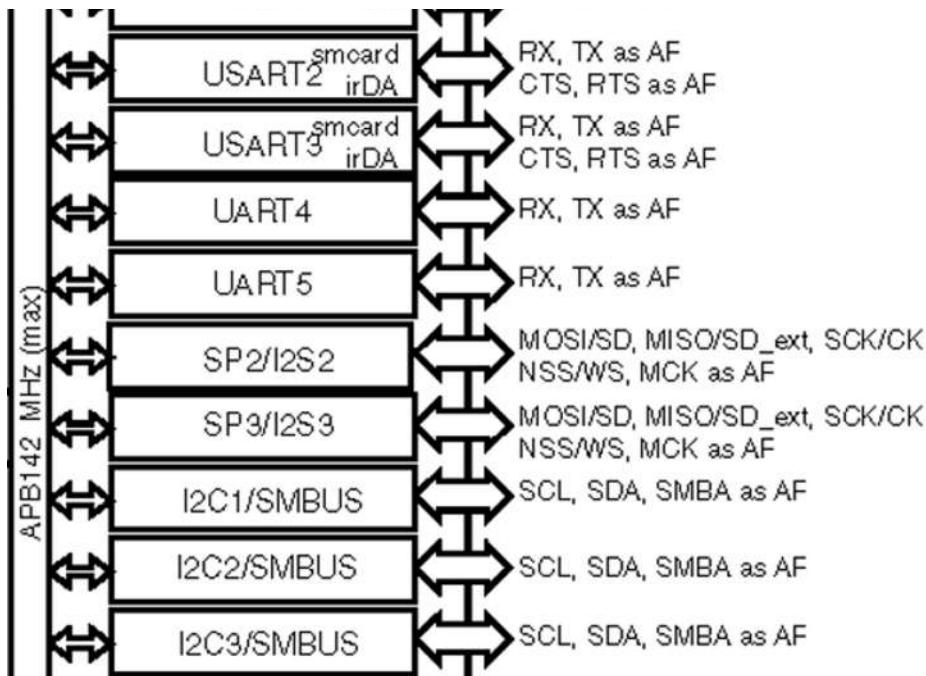
Konfigürasyon Kısımları

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
30 void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;   //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;   //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;   //PLLP 2
13    RCC->CR |= 0x01000000;        //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;      //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;      //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;      //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;      //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;       //HSERDYC
21    RCC->CIR |= 0x00800000;       //CSSC
22 }
```

- USART3 kullanacağız. Clock hattı APB1'e gidiyor.





RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	

18.biti 1 yapıyoruz.

Bit 18 **USART3EN**: USART3 clock enable

Set and cleared by software.

0: USART3 clock disabled

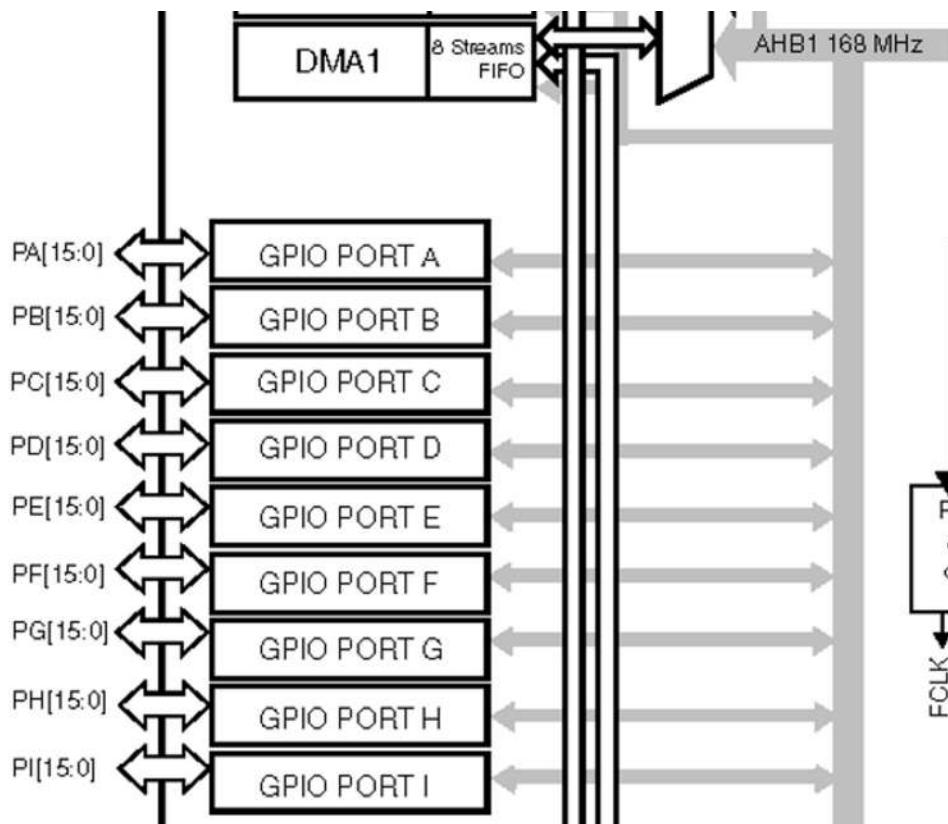
1: USART3 clock enabled

RCC->APB1ENR |= 1 << 18;

- USART3 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B portun 10. ve 11.pinine bağlı olmuş.

Port		AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
		SYS	TIM1/2	TIM3/4/5	TIM8/9/10 /11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2ext	SPI3/I2Sext /I2S3	USART1/2/3/ I2S3ext
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	-	-
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	-	-
	PB2	-	-	-	-	-	-	-	-
	PB3	JTDO/ TRACES WO	TIM2_CH2	-	-	-	SPI1_SCK	SPI3_SCK I2S3_CK	-
	PB4	NJTRST	-	TIM3_CH1	-	-	SPI1_MISO	SPI3_MISO	I2S3ext_SD
	PB5	-	-	TIM3_CH2	-	I2C1_SMB_A	SPI1_MOSI	SPI3_MOSI I2S3_SD	-
	PB6	-	-	TIM4_CH1	-	I2C1_SCL	-	-	USART1_TX
	PB7	-	-	TIM4_CH2	-	I2C1_SDA	-	-	USART1_RX
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-	-
	PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS I2S2_WS	-	-
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK I2S2_CK	-	USART3_TX
	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	-	USART3_RX
	PB12	-	TIM1_BKIN	-	-	I2C2_SMB_A	SPI2_NSS I2S2_WS	-	USART3_CK
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK I2S2_CK	-	USART3_CTS
	PB14	-	TIM1_CH2N	-	TIM8_CH2N	-	SPI2_MISO	I2S2ext_SD	USART3_RTS
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N	-	SPI2_MOSI I2S2_SD	-	-

- Portlar AHB1 kısmına gitiyor.



RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reser-ved	OTGH S ULPIE N	OTGH SEN	ETHM ACPTP EN	ETHM ACRXE N	ETHM ACTXE N	ETHMA CEN	Reserved			DMA2E N	DMA1E N	CCMDAT ARAMEN	Res.	BKPSR AMEN	Reserved	
	rw	rw	rw	rw	rw	rw				rw	rw			rw		
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Reserved		CRCE N	Reserved			GPIOE N	GPIOH EN	GPIOG EN	GPIOF N	GPIOEEN	GPIOD EN	GPIOC EN	GPIOB EN	GPIOA EN	
	rw						rw	rw	rw	rw	rw	rw	rw	rw	rw	

- Biz B portunu kullandığımızdan sadece bunu aktif ediyoruz.

Bit 1 **GPIOBEN**: IO port B clock enable

This bit is set and cleared by software.

0: IO port B clock disabled

1: IO port B clock enabled

RCC->[AHB1ENR](#) |= 0x00000002;

- B portun 10. ve 11. pinleri USART3 için kullanacağımızdan bu pinleri alternate function mode yapıyoruz.

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]	MODER14[1:0]	MODER13[1:0]	MODER12[1:0]	MODER11[1:0]	MODER10[1:0]	MODER9[1:0]	MODER8[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]	MODER6[1:0]	MODER5[1:0]	MODER4[1:0]	MODER3[1:0]	MODER2[1:0]	MODER1[1:0]	MODER0[1:0]								
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

- 00: Input (reset state)
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode

GPIOB->**MODER** |= 0x00A00000;

- Kullanacağımız çevresel birim olan USART3 kullanacağımızı belirteceğiz.

GPIO alternate function low register (GPIOx_AFRL) (x = A..I/J/K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO alternate function high register (GPIOx_AFRH)

(x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Biz 10 ve 11. pinleri kullandığımızdan high olana kullanıyoruz.

Bits 31:0 **AFRH_y**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

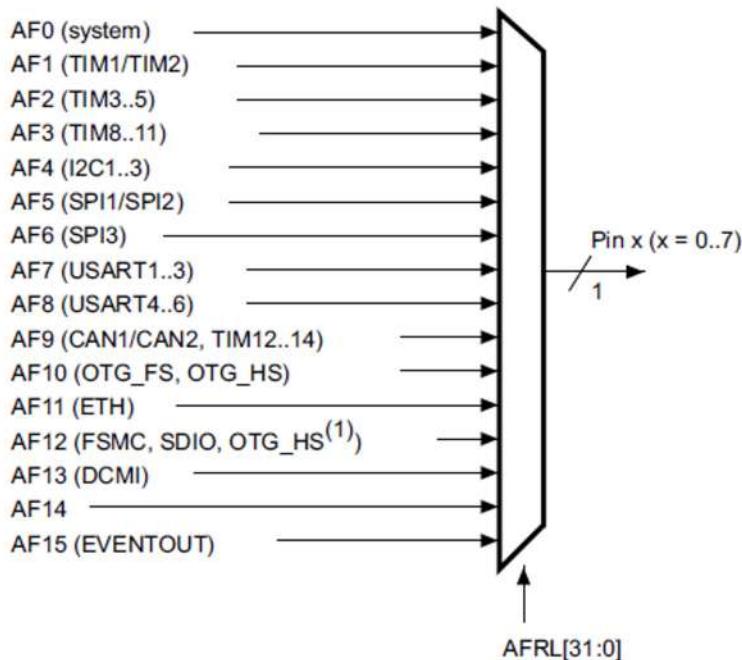
AFRH_y selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

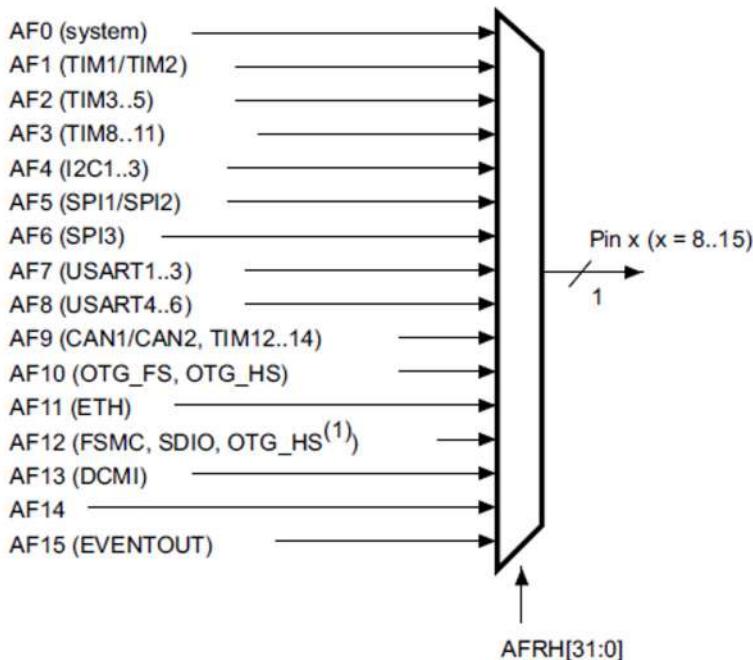
- Seçtiğimiz USART3 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

Selecting an alternate function

For pins 0 to 7, the GPIOx_AFRL[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx_AFRH[31:0] register selects the dedicated alternate function



ai17538

- USART3, AF7'de bulunuyor. Böylece pinlere AF7 için olan 0111 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunu belirtmemiz gerekiyor. High kullandığımızdan 1 yazıyoruz.

GPIOB->AFR[1] |= 7 << 8 | 7 << 12;

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```

24 void GPIO_Config(void)
25 {
26     RCC->AHB1ENR |= 0x00000002; //B clock enable
27
28     GPIOB->MODER |= 0x00A00000; //PA10, PA11 Alternate function mode
29     GPIOB->AFR[1] |= (7 << 8) | (7 << 12); //USART3 AFRH10, AFRH11
30 }

```

Baud rate register (USART_BRR)

The baud counters stop counting if the TE or RE bits are disabled respectively.

Address offset: 0x08

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DIV_Mantissa[11:0]												DIV_Fraction[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value

Bits 15:4 DIV_Mantissa[11:0]: mantissa of USARTDIV

These 12 bits define the mantissa of the USART Divider (USARTDIV)

Bits 3:0 DIV_Fraction[3:0]: fraction of USARTDIV

These 4 bits define the fraction of the USART Divider (USARTDIV). When OVER8=1, the DIV_Fraction3 bit is not considered and must be kept cleared.

- Baud rate ayarını 9600 yapmak için BRR register'a 0x1112 yazıyoruz.

USART3->BRR |= 0x1112;

Control register 1 (USART_CR1)

Address offset: 0x0C

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OVER8	Reserved	UE	M	WAKE	PCE	PS	PEIE	TXEIE	TCIE	RXNEIE	IDLEIE	TE	RE	RWU	SBK
rw	Res.	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bit 2 RE: Receiver enable

This bit enables the receiver. It is set and cleared by software.

0: Receiver is disabled

1: Receiver is enabled and begins searching for a start bit

USART3->CR1 |= (1 << 2);

Bit 3 TE: Transmitter enable

This bit enables the transmitter. It is set and cleared by software.

0: Transmitter is disabled

1: Transmitter is enabled

Note: During transmission, a "0" pulse on the TE bit ("0" followed by "1") sends a preamble (idle line) after the current word, except in smartcard mode.

When TE is set, there is a 1 bit-time delay before the transmission starts.

USART3->CR1 |= (1 << 3);

- Mesaj geldiğinde interrupt girmesi için aktif ediyoruz.

Bit 5 DYNIE: DYNIE interrupt enable

Bit 5 **RXNEIE**: RXNE interrupt enable

This bit is set and cleared by software.

0: Interrupt is inhibited

1: An USART interrupt is generated whenever ORE=1 or RXNE=1 in the USART_SF register

```
USART3->CR1 |= (1 << 5);
```

Bit 10 **PCE**: Parity control enable

This bit selects the hardware parity control (generation and detection). When the parity control is enabled, the computed parity is inserted at the MSB position (9th bit if M=1; 8th bit if M=0) and parity is checked on the received data. This bit is set and cleared by software. Once it is set, PCE is active after the current byte (in reception and in transmission).

0: Parity control disabled

1: Parity control enabled

```
USART3->CR1 |= (0 << 10);
```

Bit 12 **M**: Word length

This bit determines the word length. It is set or cleared by software.

0: 1 Start bit, 8 Data bits, n Stop bit

1: 1 Start bit, 9 Data bits, n Stop bit

Note: The M bit must not be modified during a data transfer (both transmission and reception)

```
USART3->CR1 |= (0 << 12);
```

- USART fonksiyonun en alt satırında olması gerekiyor.

Bit 13 **UE**: USART enable

When this bit is cleared, the USART prescalers and outputs are stopped and the end of the current byte transfer in order to reduce power consumption. This bit is set and cleared by software.

0: USART prescaler and outputs disabled

1: USART enabled

```
USART3->CR1 |= (1 << 13);
```

Control register 2 (USART_CR2)

Address offset: 0x10

Reset value: 0x0000 0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved																
Res.	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Res.	LINEN	STOP[1:0]	CLKEN	CPOL	CPHA	LBCL	Res.	LBDIE	LBDL	Res.					ADD[3:0]	
	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 13:12 **STOP**: STOP bits

These bits are used for programming the stop bits.

00: 1 Stop bit

01: 0.5 Stop bit

10: 2 Stop bits

11: 1.5 Stop bit

Note: The 0.5 Stop bit and 1.5 Stop bit are not available for UART4 & UART5.

```
USART3->CR2 |= (0 << 12);
```

- USART fonksiyonu için ayarlamalar bitti.

```

35 void USART_Config()
36 {
37     RCC->APB1ENR |= 1 << 18; //USART3EN
38
39     USART3->BRR |= 0x1112; //BaudRate 9600
40     USART3->CR1 |= (1 << 2); //Receiver Enable
41     USART3->CR1 |= (1 << 3); //Transmitter Enable
42     USART3->CR1 |= (1 << 5); //RXNE interrupt enable
43     USART3->CR1 |= (0 << 10); //Parity control disabled
44     USART3->CR1 |= (0 << 12); //Word length 8 Data bits
45     USART3->CR2 |= (0 << 12); //1 Stop bit
46     USART3->CR1 |= (1 << 13); //USART enable
47 }

```

Kod Kısmı

- Şimdi interrupt için fonksiyon yazacağımız. Öncelikle NVIC için fonksiyon yazıyoruz.

Status register (USART_SR)

Address offset: 0x00

Reset value: 0x0000 00C0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	
Reserved																
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
							CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
							rc_w0	rc_w0	r	rc_w0	rc_w0	r	r	r	r	

- 7.pini interrupt veriyoruz.

Bit 7 TXE: Transmit data register empty

This bit is set by hardware when the content of the TDR register has been transferred into the shift register. An interrupt is generated if the TXEIE bit =1 in the USART_CR1 register. It is cleared by a write to the USART_DR register.

0: Data is not transferred to the shift register
1: Data is transferred to the shift register

Note: This bit is used during single buffer transmission.

NVIC->ISER[1] |= (1 << 7);

- Interrupt fonksiyonu yazıyoruz. Öncelikle değişken ataması yapıyoruz.
Haberleşme durumunu Status Register ile bu değişkene yazacağız.

```

volatile int Str;
Str = USART3->SR;

```

```

49 void NVIC_Config(void)
50 {
51     NVIC->ISER[1] |= (1 << 7); //Interrupt Set Enable Register
52 }

```

- Gelen mesajları dizeye alıyoruz. Bunun için Data register kullanıyoruz.

Data register (USART_DR)

Address offset: 0x04

Reset value: 0xFFFF FFFF

Bits 31:9 Reserved, must be kept at reset value

Bits 8:0 DR[8:0]: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two

Data register (USART_DR)

Address offset: 0x04

Reset value: 0XXXX XXXX

Bits 31:9 Reserved, must be kept at reset value

Bits 8:0 DR[8:0]: Data value

Contains the Received or Transmitted data character, depending on whether it is read from or written to.

The Data register performs a double function (read and write) since it is composed of two registers, one for transmission (TDR) and one for reception (RDR)

The TDR register provides the parallel interface between the internal bus and the output shift register (see Figure 1).

The RDR register provides the parallel interface between the input shift register and the internal bus.

When transmitting with the parity enabled (PCE bit set to 1 in the USART_CR1 register), the value written in the MSB (bit 7 or bit 8 depending on the data length) has no effect because it is replaced by the parity.

When receiving with the parity enabled, the value read in the MSB bit is the received parity bit.

```
3 char Rx_Buff[100];
```

```
4 int i=0;
```

```
58     Rx_Buff[i] = USART3->DR;
59     i++;
```

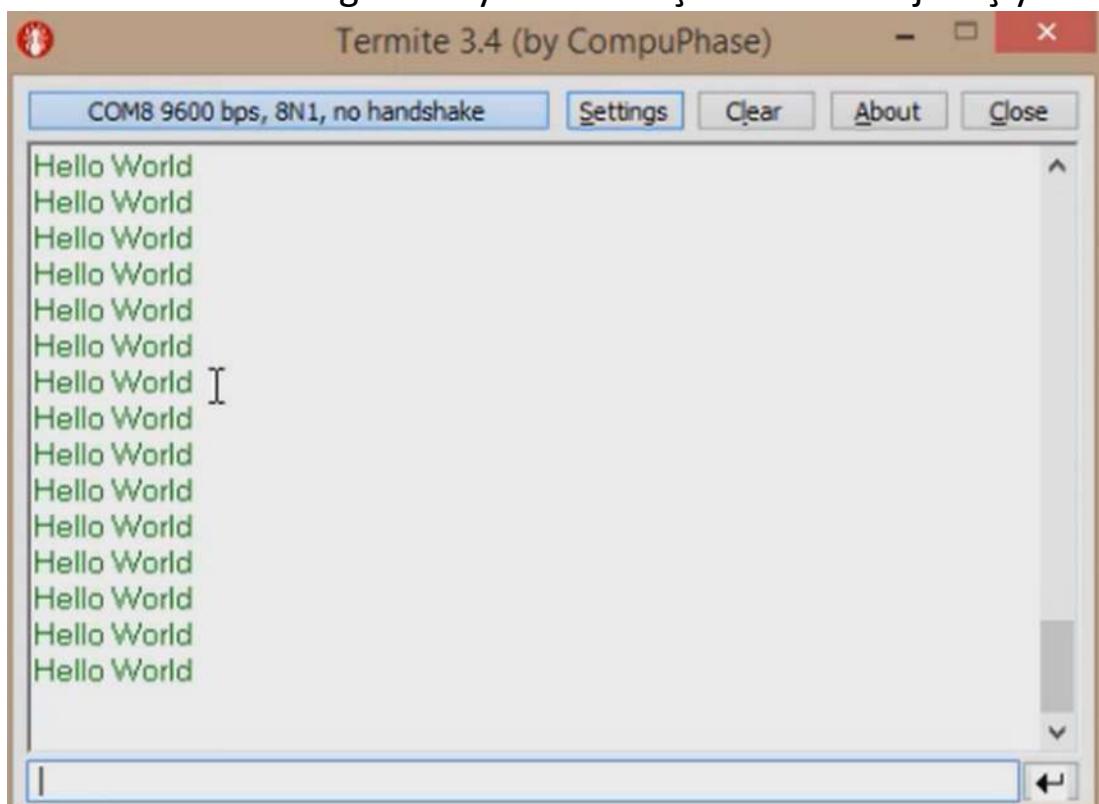
```
54 void USART3_IRQHandler()
55 {
56     volatile int Str;
57     Str = USART3->SR;
58     Rx_Buff[i] = USART3->DR;
59     i++;
60 }
```

```
62 void Send_Char(char message)
63 {
64     while(!(USART3->SR & 1 << 7));
65     USART3->DR = message;
66 }
```

```
68 void Send_Message(char *Str)
69 {
70     while(*Str)
71     {
72         Send_Char(*Str);
73         Str++;
74     }
75 }
```

```
77 int main(void)
78 {
79     RCC_Config();
80     GPIO_Config();
81     USART_Config();
82     NVIC_Config();
83
84     while (1)
85     {
86         Send_Message("Hello World \n");
87         for(int i=0; i<1000000; i++);
88     }
89 }
90 }
```

- Send_Message'a mesajımızı yazıyoruz. Mesajımızdaki harflerin yanı her dizideki elemanlarını alıyoruz ve her elemanı Send_Char'a gönderiyoruz ve burada Data register'a yazarak karşı tarafa mesaj ulaşıyor.



09_02 USART ile Led Yakma

25 Aralık 2021 Cumartesi 01:01

09_02 USART ile Led Yakma

➤ HAL

Konfigürasyon Kısmı

Kod Kısmı

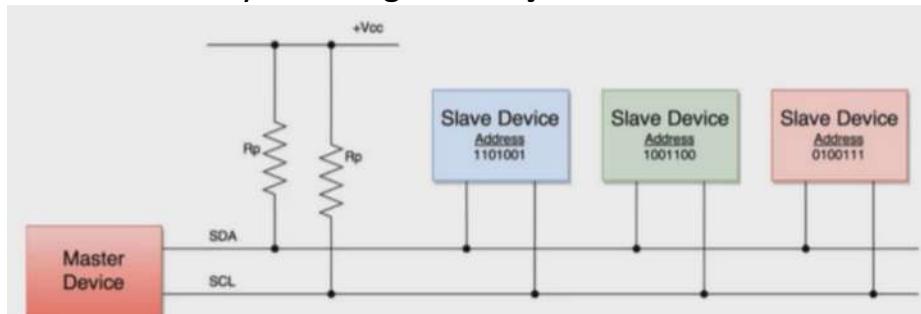
10 I2C

5 Mayıs 2021 Çarşamba 08:03

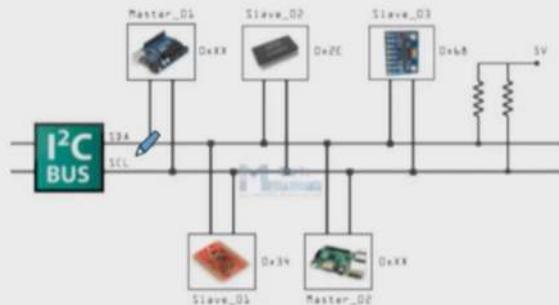
10 I2C

Giriş

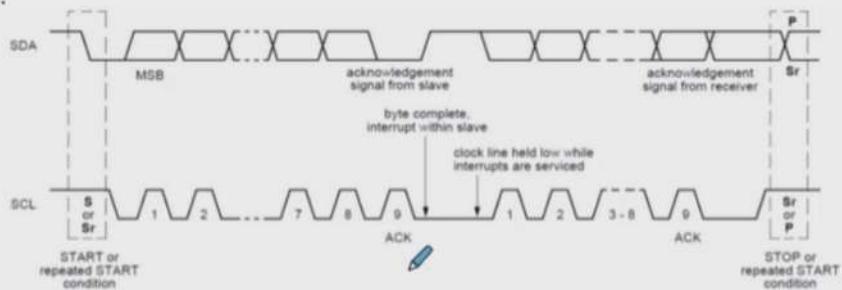
- <https://www.ercankocclar.com/2018/01/i2c-iletisim-protokolu-ve-mikroc-kutuphanesi/> linkinden ayrıntılı bilgilere ulaşabiliriz.



- I2C protokolünün geliştirilme amacı, düşük hızlı çevre birimlerinin anakartları, cep telefonları, gömülü sistemler gibi elektronik cihazlara daha az kablo ihtiyacı ile bağlanabilmesini sağlamaktır.



- I2C iletişiminde sadece iki hat vardır. Bunlar SDA (Serial Data Line) ve SCL (Serial Clock Line) hatlarıdır. Bu hatlar ayrıca pull-up direncine ihtiyaç duyarlar.
- Genellikle +5V ve +3.3V voltajlarda çalışmakla beraber, I2C protokolü daha pratik voltaj seviyelerine de izin vermektedir.



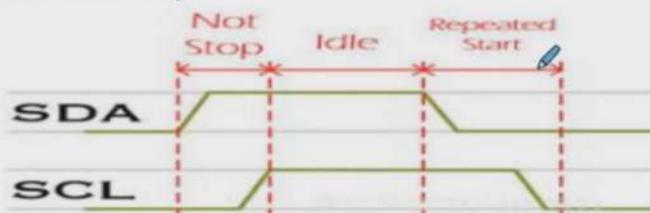
- Yukarıda görüldüğü gibi I2C iletişimini aşağıdaki sıra ile gerçekleştir:

 1. İlk olarak SDA ve SCL hatları HIGH (yüksek) konumdadırlar. Daha sonra SDA hattı master tarafından LOW(düşük) seviyeye çekilerek iletişimini başlayacağı, slave cihazlara bildirilir. (Diyagramda S ile gösterilmiştir.)
 2. Bu bildirimi alan slave cihazlar, adres bilgisini beklemeye başlarlar. Adres bilgisi slave cihazların yapısına göre 7 bit, 10 bit veya 16 bit olabilirler. Master cihaz hangi slave cihaz ile haberleşmek istiyorsa onun adres bilgisini gönderdikten sonra, okuma mı yoksa yazma mı yapacağını belirtir. Adres hangi slave cihazın ise o cihaz master ile iletişim kurmaya başlar. Adres kendisine ait olan slave cihaz, master cihaza bir ACK (Acknowledge - Kabul) biti gönderir.
 3. Veri transferi işlemi gerçekleşir. Bu transfer iki yönlü de olabilir. (Slave'den Master'a veya Master'dan Slave'e.)

- I2C veriyolu multimaster(çoklu hukmeden) bir yapıdadır. Bu sayede iletişim hattında birden fazla cihaz olabilir. Master cihazlarda bir saat sinyali ve data gönderildiği anda diğer cihazların tamamı slave moduna geçerler.
- I2C düşük bant genişliğine sahiptir ve kısa mesafelerde kullanılır.
- SDA hattı haberleşmeyi başlatıp, sonlandırır. SCL ise veri hattı konfigurasyonunu sağlar.
- I2C hattı SDA hattının lojik high seviyesinden lojik low seviyeye düşmesi ile başlar. Aynı şekilde lojik low seviyesinden lojik high seviyeye çıkması ile sonlanır. SDA hattının haberleşmeyi başlatılabilmesi için SCL hattı da high olmalıdır.



- SCL hattı lojik high seviyesinde iken SDA hattı high seviyesine çekilirse haberleşme sonlanır.
- Multimaster I2C haberleşmesinde Repeated Start komutu vardır ve sıkılıkla kullanılır. I2C haberleşmesinde 2 adet cihaz olduğunu varsayıyalım:
 - > Birinci master cihaz start komutu gönderdi ve start komutundan sonra gerekli adres bilgilerini gönderdi. Tüm bu işlemler sürecinde I2C hattı birinci master cihaz tarafından kullanıldığından dolayı I2C hattı idle(bos) durumda olmayacağıdır. Birinci cihaz stop durumu göndermeden önce haberleşmede bir değişiklik yapmak isterse Repeated Start komutunu gönderir ve böylece 1.Master cihazın slave cihaz ile I2C haberleşmesi kopmamış olur. Multimaster olmayan durumlarda Repeated Start komutunu kullanmaya gerek yoktur. Repeated Start komutu ard arda gelen start stop komutlarından oluşur.



- I2C haberleşmesinde verinin gönderildiği veya verinin alındığını doğrulamak için ACK (Acknowledge) mesajları gönderilir. I2C haberleşmesinde 1 master cihaz ve birden fazla slave cihaz olduğunu varsayıyalım. Master cihaz herhangi bir slave cihaza erişmek için start komutundan sonra ilgili slave cihazın adresini gönderir. Aynı hatta bağlı olan slave cihazların tamamı bu mesajları alır ancak sadece bu mesajı sahip olan slave cihaz Ack mesajını göndererek iletişim kurulduğu master cihaza bildirir ve Ack mesajını alan master cihaz adres bilgisinden hemen sonra veri göndermeye başlar.
- I2C veri gönderimi:

Start	Adres	W	Ack	Data	Ack	Data	Ack	Stop
-------	-------	---	-----	------	-----	------	-----	------

- Start biti "0" stop biti "1" verilerinden oluşmaktadır.
- Öncelikle master cihaz start komutunu gönderir ve devamın da haberleşmek istediği slave cihazın 7 bitlik adresini ve devamında veri göndereceğini belirttiği W (write) komutunu gönderir. Bu mesajlardan sonra eğer slave adresi ile eşleşen bir slave cihaz var ise master cihaza Ack bilgisini gönderir.
- Master cihaz Ack bilgisini aldıktan sonra 8 bitlik detayı gönderir ve tekrar slave cihazdan veri alındığına dair

10_01 I2C Kullanımı

25 Aralık 2021 Cumartesi 01:02

10_01 I2C Kullanımı

➤ SPL

Konfigürasyon Kısmı

Kod Kısmı

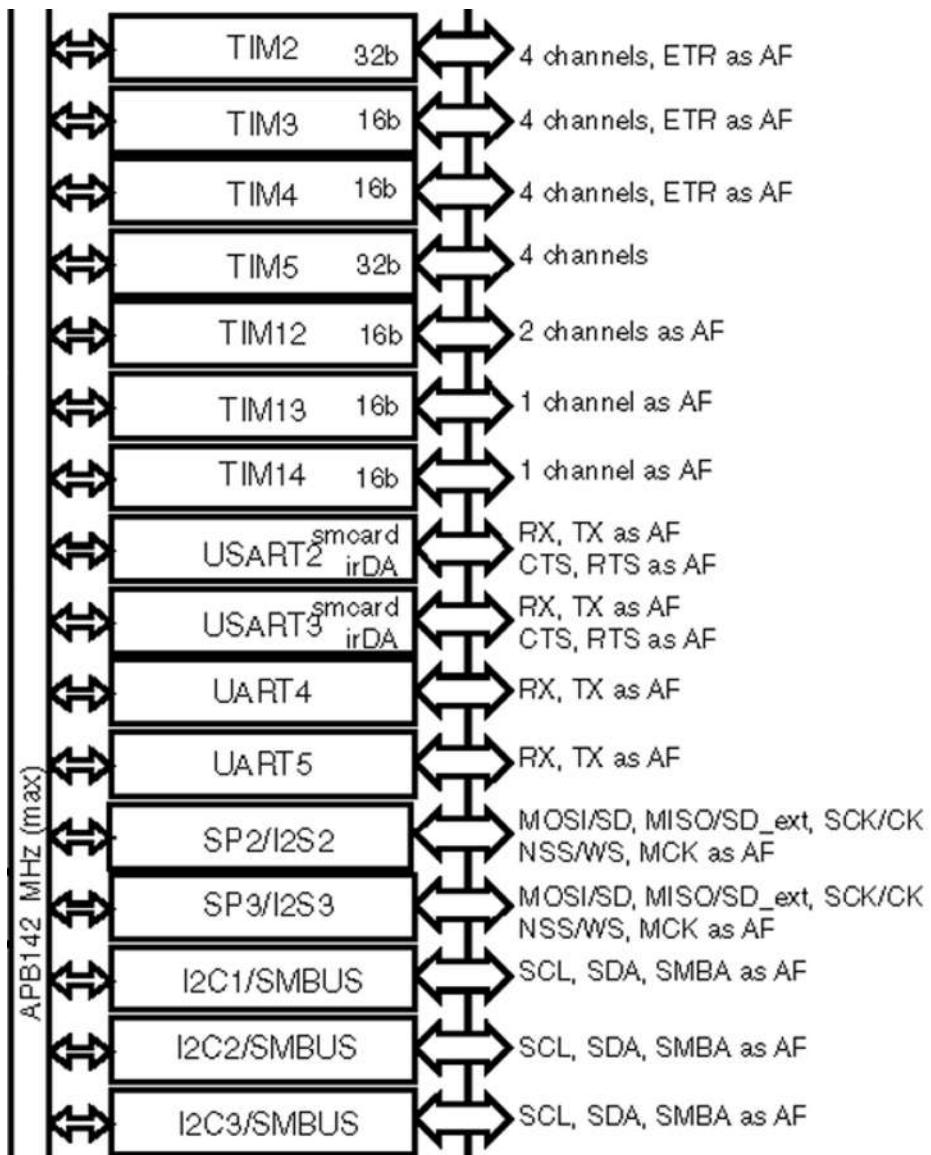
REGISTER

Konfigürasyon Kısmı

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
3@ void RCC_Config(void)
4 {
5     RCC->CR |= 0x00010000;           //HSEON
6     while(!(RCC->CR & 0x00020000)); //HSERDY
7     RCC->CR |= 0x00080000;          //CSSON
8     RCC->CFGR = 0x00000000;
9     RCC->PLLCFGR |= 0x00400000;    //PLLSRC
10    RCC->PLLCFGR |= 0x00000004;   //PLLM 4
11    RCC->PLLCFGR |= 0x00002A00;   //PLLN 168
12    RCC->PLLCFGR |= 0x00000000;   //PLLP 2
13    RCC->CR |= 0x01000000;        //PLLON
14    while(!(RCC->CR & 0x02000000)); //PLLRDY
15    RCC->CFGR |= 0x00000001;      //SW
16    while(!(RCC->CR & 0x00000001)); //SWS
17    RCC->CFGR |= 0x00000000;      //HPRE AHB 1
18    RCC->CFGR |= 0x00001400;      //PPRE1 APB1 4
19    RCC->CFGR |= 0x00008000;      //PPRE2 APB2 2
20    RCC->CIR |= 0x00080000;       //HSERDYC
21    RCC->CIR |= 0x00800000;       //CSSC
22 }
```

- I2C2 kullanacağımız. Clock hattı APB1'e gidiyor



RCC APB1 peripheral clock enable register (RCC_APB1ENR)

Address offset: 0x40

Reset value: 0x0000 0000

Access: no wait state, word, half-word and byte access.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	DAC EN	PWR EN	Reser- ved	CAN2 EN	CAN1 EN	Reser- ved	I2C3 EN	I2C2 EN	I2C1 EN	UART5 EN	UART4 EN	USART 3 EN	USART 2 EN	Reser- ved	
	rw	rw		rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SPI3 EN	SPI2 EN	Reserved	WWDG EN	Reserved	TIM14 EN	TIM13 EN	TIM12 EN	TIM7 EN	TIM6 EN	TIM5 EN	TIM4 EN	TIM3 EN	TIM2 EN		
rw	rw		rw		rw	rw	rw	rw	rw	rw	rw	rw	rw		

22.biti 1 yapıyoruz.

Bit 22 **I2C2EN**: I2C2 clock enable

This bit is set and cleared by software.

0: I2C2 clock disabled

1: I2C2 clock enabled

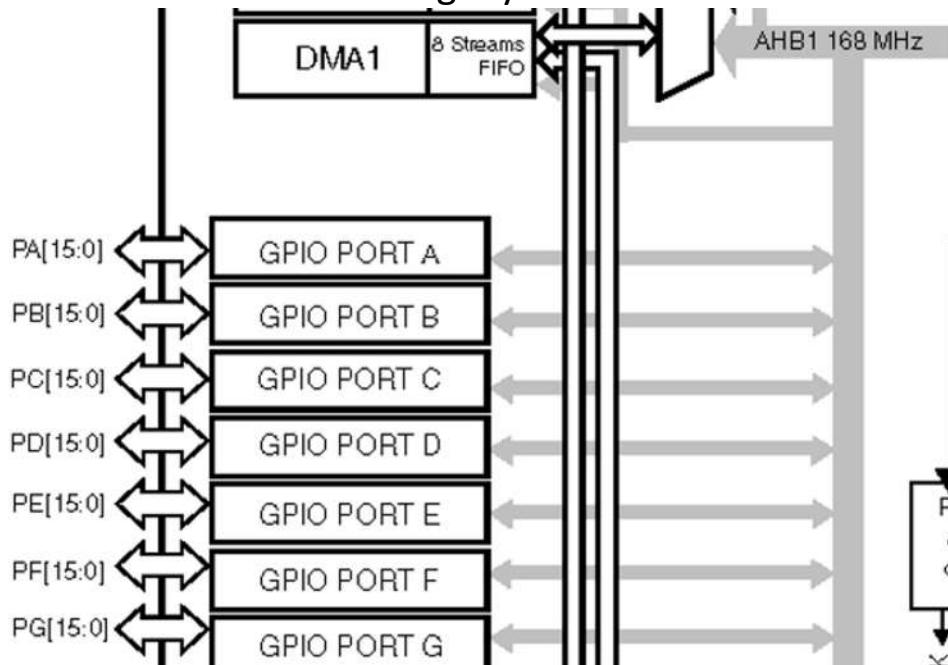
RCC->APB1ENR |= 1 << 22;

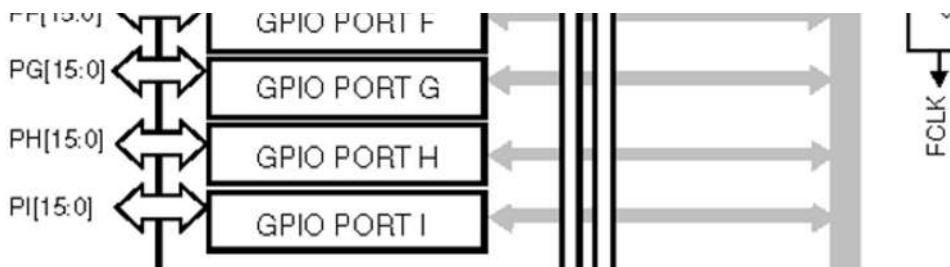
- I2C2 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B

portun 10. ve 11.pinine bağlı olmuş.

Port	AF0	AF1	AF2	AF3	AF4	AF5	AF6	AF7
	SYS	TIM1/2	TIM3/4/5	TIM8/9/10 /11	I2C1/2/3	SPI1/SPI2/ I2S2/I2S2ext	SPI3/I2Sext /I2S3	USART1/2/3/ I2S3ext
Port B	PB0	-	TIM1_CH2N	TIM3_CH3	TIM8_CH2N	-	-	-
	PB1	-	TIM1_CH3N	TIM3_CH4	TIM8_CH3N	-	-	-
	PB2	-	-	-	-	-	-	-
	PB3	JTDO/ TRACES WO	TIM2_CH2	-	-	SPI1_SCK I2S3_CK	-	-
	PB4	NJTRST	-	TIM3_CH1	-	SPI1_MISO	SPI3_MISO	I2S3ext_SD
	PB5	-	-	TIM3_CH2	I2C1_SMB_A	SPI1_MOSI	SPI3_MOSI I2S3_SD	-
	PB6	-	-	TIM4_CH1	I2C1_SCL	-	-	USART1_TX
	PB7	-	-	TIM4_CH2	I2C1_SDA	-	-	USART1_RX
	PB8	-	-	TIM4_CH3	TIM10_CH1	I2C1_SCL	-	-
	PB9	-	-	TIM4_CH4	TIM11_CH1	I2C1_SDA	SPI2_NSS I2S2_WS	-
	PB10	-	TIM2_CH3	-	-	I2C2_SCL	SPI2_SCK I2S2_CK	-
	PB11	-	TIM2_CH4	-	-	I2C2_SDA	-	USART3_RX
	PB12	-	TIM1_BKIN	-	-	I2C2_SMBA	SPI2_NSS I2S2_WS	-
	PB13	-	TIM1_CH1N	-	-	-	SPI2_SCK I2S2_CK	-
	PB14	-	TIM1_CH2N	-	TIM8_CH2N	-	SPI2_MISO	I2S2ext_SD
	PB15	RTC_REFIN	TIM1_CH3N	-	TIM8_CH3N	-	SPI2_MOSI I2S2_SD	USART3_RTS

- Portlar AHB1 kısmına gidiyor.





RCC AHB1 peripheral clock enable register (RCC_AHB1ENR)

Address offset: 0x30

Reset value: 0x0010 0000

Access: no wait state, word, half-word and byte access.

- Buton için A portunu, I2C2 için B portunu kullandığımızdan bu portları aktif ediyoruz.

Bit 0 **GPIOAEN**: IO port A clock enable

This bit is set and cleared by software.

0: IO port A clock disabled

1: IO port A clock disabled

Bit 1 **GPIOBEN**: IO port B clock enable

This bit is set and cleared by software

0: IO port B clock disabled

1: IO port B clock enabled

BCC ->AHB1ENR | = 0x00000002:

- B portun 10. ve 11. pinleri I2C için kullanacağımızdan bu pinleri alternate function mode yapıyoruz.

GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
 - 0x0000 0280 for port B
 - 0x0000 0000 for other ports

Bits 2y:2y+1 **MODERy[1:0]**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the I/O direction mode.

- 00: Input (reset state)
- 01: General purpose output mode
- 10: Alternate function mode
- 11: Analog mode

```
GPIOB->MODER |= (2 << 20) | (2 << 22) ;
```

- Kullanacağımız çevresel birim olan I2C kullanacağımızı belirteceğiz.

GPIO alternate function low register (GPIOx_AFRL) (x = A..I/J/K)

Address offset: 0x20

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRL7[3:0]				AFRL6[3:0]				AFRL5[3:0]				AFRL4[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRL3[3:0]				AFRL2[3:0]				AFRL1[3:0]				AFRL0[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

GPIO alternate function high register (GPIOx_AFRH) (x = A..I/J)

Address offset: 0x24

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
AFRH15[3:0]				AFRH14[3:0]				AFRH13[3:0]				AFRH12[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
AFRH11[3:0]				AFRH10[3:0]				AFRH9[3:0]				AFRH8[3:0]			
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

- Biz 10 ve 11. pinleri kullandığımızdan high olanı kullanıyoruz.

Bits 31:0 **AFRH_y**: Alternate function selection for port x bit y (y = 8..15)

These bits are written by software to configure alternate function I/Os

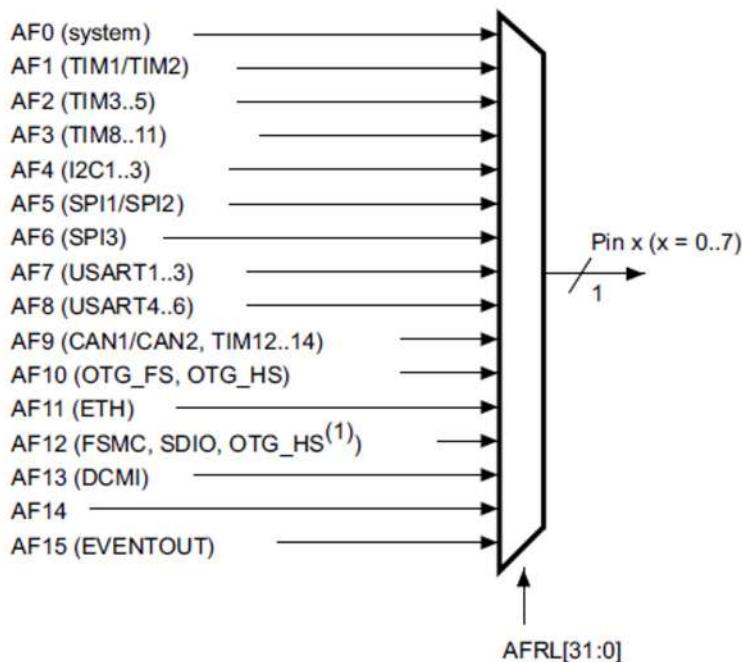
AFRH_y selection:

0000: AF0	1000: AF8
0001: AF1	1001: AF9
0010: AF2	1010: AF10
0011: AF3	1011: AF11
0100: AF4	1100: AF12
0101: AF5	1101: AF13
0110: AF6	1110: AF14
0111: AF7	1111: AF15

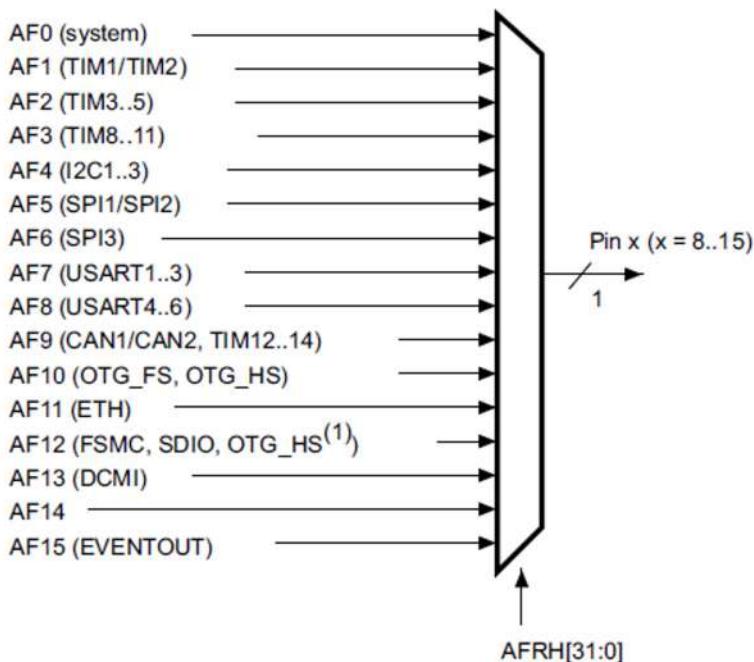
- Seçtiğimiz I2C2 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitabıçığına bakıyoruz.

Selecting an alternate function

For pins 0 to 7, the GPIOx_AFRL[31:0] register selects the dedicated alternate function



For pins 8 to 15, the GPIOx_AFRH[31:0] register selects the dedicated alternate function



ai17538

- I2C, AF7'de bulunuyor. Böylece pinlere AF4 için olan 0100 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunu belirtmemiz gerekiyor. High kullandığımızdan 1 yazıyoruz.

`GPIOB->AFR[1] |= (4 << 8) | (4 << 12);`

GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 15:0 **OTy**: Port x configuration bits (y = 0..15)

These bits are written by software to configure the output type of the I/O port.

0: Output push-pull (reset state)

1: Output open-drain

`GPIOB->OTYPER |= (1 << 10) | (1 << 11);`

- GPIO için yazdığımız fonksiyon aşağıdaki gibidir.

```
24 void GPIO_Config()
25 {
26     RCC->AHB1ENR |= (3 << 0); //A, B clock enable
27
28     GPIOB->MODER |= (2 << 20) | (2 << 22); //PB10, PB11 Alternate function mode
29     GPIOB->AFR[1] |= (4 << 8) | (4 << 12); //I2C2 AFRH10, AFRH11
30     GPIOB->OTYPER |= (1 << 10) | (1 << 11); //Output open-drain
31 }
```

I²C Control register 2 (I2C_CR2)

Address offset: 0x04

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	LAST		DMAEN	ITBUFEN	ITEVTEN	ITERREN	Reserved	FREQ[5:0]							
	rw	rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw	rw	rw

Bits 5:0 **FREQ[5:0]**: Peripheral clock frequency

The FREQ bits must be configured with the APB clock frequency value (I2C peripheral connected to APB). The FREQ field is used by the peripheral to generate data setup and hold times compliant with the I2C specifications. The minimum allowed frequency is 2 MHz, the maximum frequency is limited by the maximum APB frequency and cannot exceed 50 MHz (peripheral intrinsic maximum limit).

0b000000: Not allowed

0b000001: Not allowed

0b000010: 2 MHz

...

0b110010: 50 MHz

Higher than 0b100100: Not allowed

`I2C2->CR2 |= 0x0008;`

I²C Clock control register (I²C_CCR)

Address offset: 0x1C

Reset value: 0x0000

f_{PCLK1} must be at least 2 MHz to achieve Sm mode I²C frequencies. It must be at least 4 MHz to achieve Fm mode I²C frequencies. It must be a multiple of 10MHz to reach the 400 kHz maximum I²C Fm mode clock.

The CCR register must be configured only when the I²C is disabled (PE = 0).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
F/S	DUTY	Reserved	CCR[11:0]													
rw	rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 11:0 CCR[11:0]: Clock control register in Fm/Sm mode (Master mode)

Controls the SCL clock in master mode.

Sm mode or SMBus:

$$T_{high} = CCR * T_{PCLK1}$$

$$T_{low} = CCR * T_{PCLK1}$$

Fm mode:

If DUTY = 0:

$$T_{high} = CCR * T_{PCLK1}$$

$$T_{low} = 2 * CCR * T_{PCLK1}$$

If DUTY = 1: (to reach 400 kHz)

$$T_{high} = 9 * CCR * T_{PCLK1}$$

$$T_{low} = 16 * CCR * T_{PCLK1}$$

For instance: in Sm mode, to generate a 100 kHz SCL frequency:

If FREQR = 08, $T_{PCLK1} = 125$ ns so CCR must be programmed with 0x28

(0x28 => 40d x 125 ns = 5000 ns.)

Note: The minimum allowed value is 0x04, except in FAST DUTY mode where the minimum allowed value is 0x01

$t_{high} = t_f(SCL) + t_w(SCLH)$. See device datasheet for the definitions of parameters.

$t_{low} = t_f(SCL) + t_w(SCLL)$. See device datasheet for the definitions of parameters.

I²C communication speed, $f_{SCL} \sim 1/(t_{high} + t_{low})$. The real frequency may differ due to the analog noise filter input delay.

The CCR register must be configured only when the I²C is disabled (PE = 0).

- 100 kHz çalışmak için 0x28 yazınız demis.

I²C2->[CR2](#) |= 0x0008;

I²C TRISE register (I²C_TRISE)

Address offset: 0x20

Reset value: 0x0002

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved										TRISE[5:0]					
										rw	rw	rw	rw	rw	rw

Bits 15:6 Reserved, must be kept at reset value

Bits 5:0 **TRISE[5:0]**: Maximum rise time in Fm/Sm mode (Master mode)

These bits should provide the maximum duration of the SCL feedback loop in master mode. The purpose is to keep a stable SCL frequency whatever the SCL rising edge duration. These bits must be programmed with the maximum SCL rise time given in the I²C bus specification, incremented by 1.

For instance: in Sm mode, the maximum allowed SCL rise time is 1000 ns.

If, in the I2C_CR2 register, the value of FREQ[5:0] bits is equal to 0x08 and T_{PCLK1} = 125 ns therefore the TRISE[5:0] bits must be programmed with 09h.

(1000 ns / 125 ns = 8 + 1)

The filter value can also be added to TRISE[5:0].

If the result is not an integer, TRISE[5:0] must be programmed with the integer part, in order to respect the t_{HIGH} parameter.

Note: TRISE[5:0] must be configured only when the I2C is disabled (PE = 0).

- Geri besleme döngüsünün maksimum süresi için I2C_CR2'nin FREQ[5:0] bitlerine 0x08 yazdığımızdan TRISE[5:0] bitlerine 0x09 yazmalıyız.

I2C2->**TRISE** |= 0x09;

I²C Control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENGC	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bit 0 **PE**: Peripheral enable

0: Peripheral disable

1: Peripheral enable

Note: If this bit is reset while a communication is on going, the peripheral is disabled at the end of the current communication, when back to IDLE state.

All bit resets due to PE=0 occur at the end of the communication.

In master mode, this bit must not be reset before the end of the communication.

- I2C2 aktif edildi.

I2C2->**CR1** |= (1 << 0);

I2C için yazdığımız fonksiyon aşağıdaki gibidir.

```
33 void I2C_Config()
34 {
35     RCC->APB1ENR |= 1 << 22;           //I2CEN
36
37     I2C2->CR2 |= 0x0008;                //8MHz
38     I2C2->CCR |= 0x0028;                //100kHz
39     I2C2->TRISE |= 0x09;                //Maximum rise time
40     I2C2->CR1 |= (1 << 0);            //Peripheral enable
41 }
```

Kod Kısmı

- I2C için Write ve Read fonksiyonlarını yazmaya başlıyoruz.
- I2C_Write fonksiyonuna adres ve data olmak üzere 2 parametre yazıyoruz.

Bit 8 **START**: Start generation

This bit is set and cleared by software and cleared by hardware when start is sent or PE=0.

In Master Mode:

0: No Start generation

1: Repeated start generation

In Slave mode:

0: No Start generation

1: Start generation when the bus is free

- Start yolluyorum.

```
I2C2->CR1 |= (1 << 8);
```

I²C Status register 1 (I2C_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

- Yollandığım startı kontrol ediyorum.

Bit 0 **SB**: Start bit (Master mode)

0: No Start condition

1: Start condition generated.

– Set when a Start condition generated.

– Cleared by software by reading the SR1 register followed by writing the DR register, or by hardware when PE=0

- While döngüsü 1 olduğu sürece çalışır.
- SB biti 1 olana dek 0 olacağından ve döngünün çalışabilmesi için başına "!" işaretini koyarak tersliyoruz.

0.bit 1 olduğunda döngüden çıkışacaktır.

```
while(!(I2C2->SR1 & (1 << 0)));
```

- Slave tarafına adresini yolluyoruz. Bizim kullandığımız cihazın adresi 0x4E'dir.

I²C Data register (I2C_DR)

Address offset: 0x10

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved								DR[7:0]							
								rw	rw	rw	rw	rw	rw	rw	rw

Bits 7:0 **DR[7:0]** 8-bit data register

Byte received or to be transmitted to the bus.

– Transmitter mode: Byte transmission starts automatically when a byte is written in the DR register. A continuous transmit stream can be maintained if the next data to be transmitted is put in DR once the transmission is started (TxE=1)

– Receiver mode: Received byte is copied into DR (RxNE=1). A continuous transmit stream can be maintained if DR is read before the next data byte is received (RxNE=1).

Note: In slave mode, the address is not copied into DR.

Write collision is not managed (DR can be written if TxE=0).

If an ARLO event occurs on ACK pulse, the received byte is not copied into DR and so cannot be read.

```
I2C2->DR = 0x4E;
```

- Slave adresin gönderimini beklememiz lazım. Bunun için kontrol yapıyoruz.

I²C Status register 1 (I2C_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 1 **ADDR**: Address sent (master mode)/matched (slave mode)

This bit is cleared by software reading SR1 register followed reading SR2, or by hardware when PE=0.

Address matched (Slave)

0: Address mismatched or not received.

1: Received address matched.

- Set by hardware as soon as the received slave address matched with the OAR registers content or a general call or a SMBus Device Default Address or SMBus Host or SMBus Alert is recognized. (when enabled depending on configuration).

Note: In slave mode, it is recommended to perform the complete clearing sequence (READ SR1 then READ SR2) after ADDR is set. Refer to [Figure 242](#).

Address sent (Master)

0: No end of address transmission

1: End of address transmission

- For 10-bit addressing, the bit is set after the ACK of the 2nd byte.
- For 7-bit addressing, the bit is set after the ACK of the byte.

Note: ADDR is not set after a NACK reception

```
while(!(I2C2->SR1 & (1 << 1)));
```

- Master oldu mu onu kontrol ediyoruz.

I²C Status register 2 (I2C_SR2)

Address offset: 0x18

Reset value: 0x0000

Reading I2C_SR2 after reading I2C_SR1 clears the ADDR flag, even if the ADDR flag was set after reading I2C_SR1. Consequently, I2C_SR2 must be read only when ADDR is found set in I2C_SR1 or when the STOPF bit is cleared.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PEC[7:0]								DUALF	SMB HOST	SMBDE FAULT	GEN CALL	Res.	TRA	BUSY	MSL
r	r	r	r	r	r	r	r	r	r	r	r		r	r	r

Bit 0 **MSL**: Master/slave

0: Slave Mode

1: Master Mode

- Set by hardware as soon as the interface is in Master mode (SB=1).
- Cleared by hardware after detecting a Stop condition on the bus or a loss of arbitration (ARLO=1), or by hardware when PE=0.

```
while(!(I2C2->SR2 & (1 << 0)));
```

- TxE'nin boş olmasını bekliyoruz.

I²C Status register 1 (I2C_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 7 **TxE**: Data register empty (transmitters)

0: Data register not empty

1: Data register empty

- Set when DR is empty in transmission. TxE is not set during address phase.

- Cleared by software writing to the DR register or by hardware after a start or a stop condition or when PE=0.

TxE is not set if either a NACK is received, or if next byte to be transmitted is PEC (PEC=1)

Note: TxE is not cleared by writing the first data being transmitted, or by writing data when BTF is set, as in both cases the data register is still empty.

```
while(!(I2C2->SR2 & (1 << 7)));
```

```
I2C2->DR = data;
```

I²C Status register 1 (I2C_SR1)

Address offset: 0x14

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SMB ALERT	TIME OUT	Res.	PEC ERR	OVR	AF	ARLO	BERR	TxE	RxNE	Res.	STOPF	ADD10	BTF	ADDR	SB
rc_w0	rc_w0		rc_w0	rc_w0	rc_w0	rc_w0	rc_w0	r	r		r	r	r	r	r

Bit 2 **BTF**: Byte transfer finished

0: Data byte transfer not done

1: Data byte transfer succeeded

- Set by hardware when NOSTRETCH=0 and:

- In reception when a new byte is received (including ACK pulse) and DR has not been read yet (RxNE=1).

- In transmission when a new byte should be sent and DR has not been written yet (TxE=1).

- Cleared by software by either a read or write in the DR register or by hardware after a start or a stop condition in transmission or when PE=0.

Note: The BTF bit is not set after a NACK reception

The BTF bit is not set if next byte to be transmitted is the PEC (TRA=1 in I2C_SR2 register and PEC=1 in I2C_CR1 register)

```
while(!(I2C2->SR2 & (1 << 2)));
```

I²C Control register 1 (I2C_CR1)

Address offset: 0x00

Reset value: 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
SWRST	Res.	ALERT	PEC	POS	ACK	STOP	START	NO STRETCH	ENG C	ENPEC	ENARP	SMB TYPE	Res.	SMBUS	PE
rw		rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw		rw	rw

Bit 9 **STOP**: Stop generation

The bit is set and cleared by software, cleared by hardware when a Stop condition is detected, set by hardware when a timeout error is detected.

In Master Mode:

0: No Stop generation.

1: Stop generation after the current byte transfer or after the current Start condition is sent.

In Slave mode:

0: No Stop generation.

1: Release the SCL and SDA lines after the current byte transfer.

```
I2C2->CR1 |= (1 << 9);
```

- I²C için yazdığımız Write fonksiyon aşağıdaki gibidir.

```

43 void I2C_Write(uint8_t adress, uint8_t data)
44 {
45     I2C2->CR1 |= (1 << 8);           //Start generation
46     while(!(I2C2->SR1 & (1 << 0)));   //Start bit
47     I2C2->DR = 0x4E;                  //Slave address
48     while(!(I2C2->SR1 & (1 << 1)));   //Received address matched
49     while(!(I2C2->SR2 & (1 << 0)));   //Master Mode
50     //I2C2->DR = adress;
51     while(!(I2C2->SR2 & (1 << 0)));   //Master Mode
52     while(!(I2C2->SR2 & (1 << 7)));   //Data register empty
53     I2C2->DR = data;                 //Data byte transfer succeeded
54     while(!(I2C2->SR2 & (1 << 2)));   //Stop generation
55     I2C2->CR1 |= (1 << 9);
56 }

```

- Her butona bastığımızda cihaza sırayla adres yolluyor.

```

65 int main(void)
66 {
67     RCC_Config();
68     GPIO_Config();
69     I2C_Config();
70
71     while (1)
72     {
73         if(GPIOA->IDR & 0x00000001)
74         {
75             i++;
76             delay(6300000);
77         }
78
79         switch(i)
80         {
81             case 0:
82                 I2C_Write(m_address, 0x00);
83                 break;
84             case 1:
85                 I2C_Write(m_address, 0x01);
86                 break;
87             case 2:
88                 I2C_Write(m_address, 0x02);
89                 break;
90             case 3:
91                 I2C_Write(m_address, 0x04);
92                 break;
93             case 4:
94                 I2C_Write(m_address, 0x08);
95                 break;
96             case 5:
97                 I2C_Write(m_address, 0x10);
98                 break;
99             case 6:
100                I2C_Write(m_address, 0x20);
101                break;
102            case 7:
103                I2C_Write(m_address, 0x40);
104                break;
105            case 8:
106                I2C_Write(m_address, 0x80);
107                break;
108            default:
109                i=0;
110                break;
111        }
112    }

```

```
110         break;
111     }
112 }
113 }
114 }
```

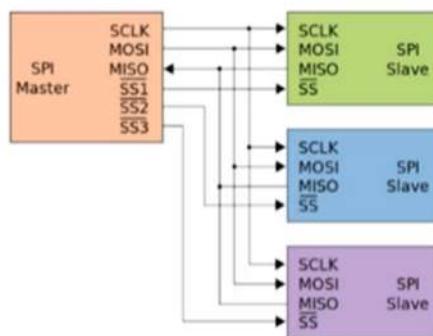
11 SPI

5 Mayıs 2021 Çarşamba 08:03

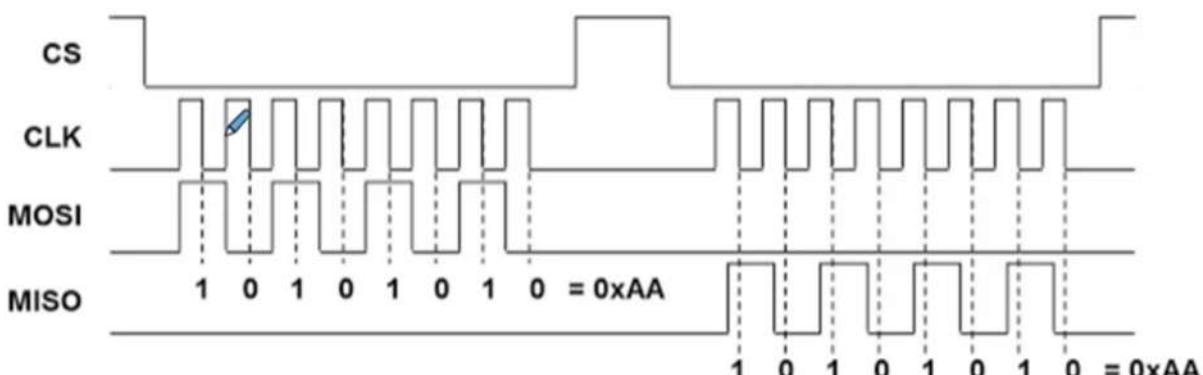
11 SPI

Giriş

- SPI (Serial Peripheral Interface), STM32F4'ün desteklediği senkron seri haberleşme türlerinden biridir. Özellik ve kullanım olarak I2C'ye benzer. Bir STM32F4'ün diğer STM32F4 veya sensörlerle kısa mesafede haberleşmesini sağlar. SPI protokolünde de I2C'de olduğu gibi bir adet Master cihaz bulunur. Bu cihaz hatta bağlı çevresel cihazları kontrol eder.



- Çevresel cihazlarla veya diğer mikrodenetleyicilerle veri transferi sağlayan yazılım/donanım tabanlı seri iletişim protokolüdür. Bu haberleşme şekli karşılıklı iki tarafın clocklarının senkronize bir şekilde çalışmasıyla data iletişimini sağlamaktadır.
- Bunlar;
 - > MOSI (Master Output Slave Input): Master cihazdan Slave cihaza sinyal taşıyan hat
 - > MISO (Master Input Slave Output): Slave cihazdan Master cihaza sinyal taşıyan hat
- SCLK sinyali senkrol olarak bu iki MOSI ve MISO sinalının taşınmasında kullanılır. Bu sinyal sadece master cihaz tarafından üretilir.
- SPI'da veri transfer hızı I2C veri yolundan daha hızlıdır. Slave cihaz donanımsal olarak seçildiği için (SS veya CS pini üzerinden) slave cihaza I2C veri iletişimindeki gibi adres gönderilmez.
- Fakat birden fazla slave cihazın SPI veri yoluna bağlanması için birden fazla SS veya CS pini kullanılır.
- SPI iletişiminde, önce çalışmak istenilen slave cihazın bağlı olduğu SS pini seçilir.
- Master tarafından verinin en öncelikli kısmından (MSB) itibaren MOSI hattı üzerinden slave cihaz tarafına her clock pulsunda bir bit olmak üzere tüm veri gönderilir.



11_01 SPI Kullanımı

14 Mayıs 2022 Cumartesi 11:49

11_01 SPI Kullanımı

- <https://deepbluemedded.com/stm32-spi-tutorial/>, <https://www.youtube.com/watch?v=E5Q7hP2Ni2Q>,
<https://www.youtube.com/watch?v=LHsxzm7-NMI>, <https://www.youtube.com/watch?v=9EKCEXNP1i8>,
<https://youtu.be/l1G7IgulKh0>, https://youtu.be/-2kH7ei_Ljw, <https://youtu.be/BvUMpjEoctY> linklerinden konu hakkındaki uygulamaları inceleyebiliriz.

➤ HAL

Master

Konfigürasyon Kısmı

Kod Kısmı

Slave

Konfigürasyon Kısmı

Kod Kısmı

12 RNG

5 Mayıs 2021 Çarşamba 08:03

12 RNG

12_01 RNG Kullanımı

25 Aralık 2021 Cumartesi 01:02

12_01 RNG Kullanımı

➤ HAL

Konfigürasyon Kısmı

Kod Kısmı

➤ SPL

Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

Kod Kısmı

13 Flash Memory

5 Mayıs 2021 Çarşamba 08:03

13 Flash Memory

13_01 Flash Memory Kullanımı

25 Aralık 2021 Cumartesi 01:02

01 Flash Memory Kullanımı

➤ HAL

Konfigürasyon Kısmı

Kod Kısmı

➤ SPL

Konfigürasyon Kısmı

Kod Kısmı

➤ REGISTER

Konfigürasyon Kısmı

Kod Kısmı

14 Örnek Projeler

23 Kasım 2021 Salı 21:12

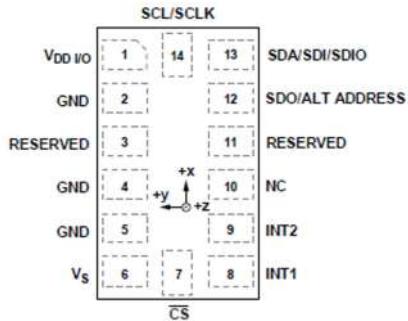
14 Örnek Projeler

14_01 ADXL345

9 Ocak 2022 Pazar 19:21

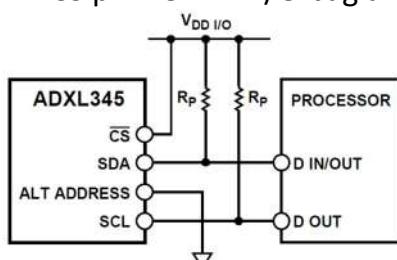
14_01 ADXL345

- Kart hakkında bilgi edinmek için <https://learn.adafruit.com/adxl345-digital-accelerometer> linki okunabilir.
- <https://controllerstech.com/adxl345-accelerometer-using-stm32/>, <https://youtu.be/m2iDrosoc2g> ve <https://youtu.be/KMhbV1p3MWk> örnek uygulamayı inceleyebiliriz.
- https://www.nxp.com/files-static/sensors/doc/app_note/AN3461.pdf ile üç eksenli ivmeölçer ile eğim algılama hakkında fikir edinebiliriz.
- Kütüphane için <https://github.com/CanGuveren/ADXL345-STM32> bu linki kaynak alabiliriz.



Pin No.	Mnemonic	Description
1	V _{DD I/O}	Digital Interface Supply Voltage.
2	GND	Must be connected to ground.
3	Reserved	Reserved. This pin must be connected to V _S or left open.
4	GND	Must be connected to ground.
5	GND	Must be connected to ground.
6	V _S	Supply Voltage.
7	CS	Chip Select.
8	INT1	Interrupt 1 Output.
9	INT2	Interrupt 2 Output.
10	NC	Not Internally Connected.
11	Reserved	Reserved. This pin must be connected to ground or left open.
12	SDO/ALT ADDRESS	Serial Data Output/Alternate I ² C Address Select.
13	SDA/SDI/SDIO	Serial Data (I ² C)/Serial Data Input (SPI 4-Wire)/Serial Data Input and Output (SPI 3-Wire).
14	SCL/SCLK	Serial Communications Clock.

- CS pinine VDD I/O bağlandığında yani besleme hattı çekilirse ADXL345 sensörü I²C modundadır.



- Standart (100kHz) ve Hızlı (400kHz) veri aktarım modlarını destekler.
- Tek veya çok byte okuma ve yazma desteklenir.

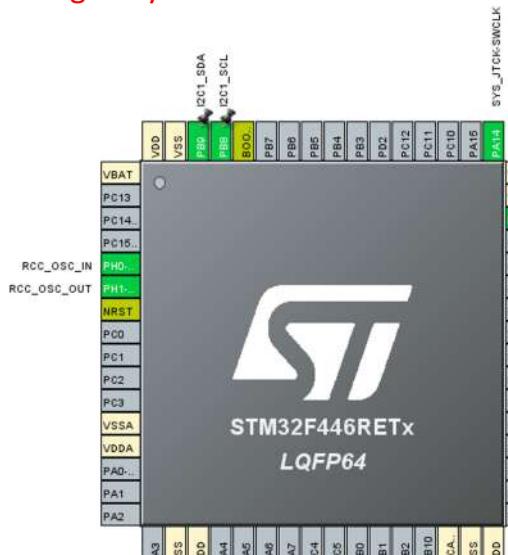
SINGLE-BYTE WRITE									
MASTER	START	SLAVE ADDRESS + WRITE	REGISTER ADDRESS	DATA	STOP				
SLAVE			ACK	ACK	ACK				
MULTIPLE-BYTE WRITE									
MASTER	START	SLAVE ADDRESS + WRITE	REGISTER ADDRESS	DATA	DATA	DATA	STOP		
SLAVE			ACK	ACK	ACK	ACK	ACK		
SINGLE-BYTE READ									
MASTER	START	SLAVE ADDRESS + WRITE	REGISTER ADDRESS	START ¹	SLAVE ADDRESS + READ	NACK	STOP		
SLAVE			ACK	ACK	ACK	DATA			
MULTIPLE-BYTE READ									
MASTER	START	SLAVE ADDRESS + WRITE	REGISTER ADDRESS	START ¹	SLAVE ADDRESS + READ	ACK		NACK	STOP
SLAVE			ACK	ACK	ACK	DATA		DATA	

- ALT ADDRESS pini high olduğunda 7 bit I²C adresi 0x1D'dir ve ardından R/W biti gelir. Yazma için 0x3A, okuma için 0x3B kullanılır.
- ALT ADDRESS pini topraklanarak alternatif I²C adresi 0x53 yapılır ve ardından R/W biti gelir. Yazma için 0xA6, okuma için 0xA7 kullanılır.
- Uygun çalışma için pull-up direnci gerekebilir. Bununla ilgili ayrıntılarla

<https://antrak.org.tr/genel/i2c-bus-veri-ileti%C5%9Fim-standartlar%C4%B1/> linkinden öğrenebiliriz.

➤ Üç Eksen Değer Okuma

Konfigürasyon Kısımlı



Pin N...	Signal on...	GPIO out...	GPIO mo...	GPIO Pull...	Maximum...	User Label	Modified
PB8	I2C1_SCL	n/a	Alternate ...	Pull-up	Very High	<input type="checkbox"/>	<input type="checkbox"/>
PB9	I2C1_SDA	n/a	Alternate ...	Pull-up	Very High	<input type="checkbox"/>	<input type="checkbox"/>

- Connectivity başlığı altında I2C seçip Mode ekranından I2C seçiyoruz.

I2C I2C

- I2C için Speed Mode olarak Fast Mode seçimi yapıyoruz.

Master Features

I2C Speed Mode	Fast Mode
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2

Slave Features

Clock No Stretch Mode	Disabled
Primary Address Length s...	7-bit
Dual Address Acknowledg.	Disabled
Primary slave address	0
General Call address dete..	Disabled

Kod Kısmı

```

45 /* USER CODE BEGIN PV */
46 uint8_t i;
47 /* USER CODE END PV */

49 /* Private function prototypes */
50 void SystemClock_Config(void);
51 static void MX_GPIO_Init(void);
52 static void MX_I2C1_Init(void);
53 /* USER CODE BEGIN PFP */
54 void Scan_I2C_Adress();
55 /* USER CODE END PFP */

88 /* Initialize all configured peripherals */
89 MX_GPIO_Init();
90 MX_I2C1_Init();
91 /* USER CODE BEGIN 2 */
92 Scan_I2C_Adress();
93 /* USER CODE END 2 */

```

```

208 /* USER CODE BEGIN 4 */
209 void Scan_I2C_Adress()
210 {
211     for(int i=0; i<=255; i++)
212     {
213         if(HAL_I2C_IsDeviceReady(&hi2c1, i, 1, 10) == HAL_OK)
214             break;
215     }
216 }
217 /* USER CODE END 4 */

```

3420 HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)

- HAL_I2C_IsDeviceReady() fonksiyonu ile hedef cihazın iletişim için hazır olup olmadığını kontrol ederiz.
- Bu fonksiyona hal_i2c.c dosyasından ulaşırız.
- Trials deneme sayısını ifade eder.
- Yolladığımız i değişkeni eğer eşleşirse break komutu ile if yapısından çıkar.
- 92.satır ve 214.satırda break point koyuyoruz ve debug tıklıyoruz. Ardından i değişkenimizi Watch Expression kısmına ekliyoruz. Tarama işlemi bitirdiğinde bize 166 sayısını gösteriyor yani A6 adresi olduğunu gösterir.

HEX	A6
DEC	166
OCT	246
BIN	1010 0110

- Aslında bulduğum adres, 0x53 adresin 1 bit kaydırılmış halidir. 0x53 adresi 0101 0011 iken 0xA6 adresi 1010 0110'dır.
- 0x53 << 1 tanımı adresi 1 bit sola kaydır demektir.
- Önce bulduğum I2C adresini tanımlıyorum.

```

23/* Private includes -----
24/* USER CODE BEGIN Includes */
25#define ADXL345_I2C_SLAVE_ADDRESS 0xA6 //0x53<<1
26/* USER CODE END Includes */

```

- İlk kullanıma hazır hale getiriyoruz.
- Datasheet kısmından sensörün register harmasını inceliyoruz.

Address		Name	Type	Reset Value	Description
Hex	Dec				
0x00	0	DEVID	R	11100101	Device ID. Reserved. Do not access.
0x01 to 0x01C	1 to 28	Reserved			
0x1D	29	THRESH_TAP	R/W	00000000	Tap threshold.
0x1E	30	OFSX	R/W	00000000	X-axis offset.
0x1F	31	OFSY	R/W	00000000	Y-axis offset.
0x20	32	OFSZ	R/W	00000000	Z-axis offset.
0x21	33	DUR	R/W	00000000	Tap duration.
0x22	34	Latent	R/W	00000000	Tap latency.
0x23	35	Window	R/W	00000000	Tap window.
0x24	36	THRESH_ACT	R/W	00000000	Activity threshold.
0x25	37	THRESH_INACT	R/W	00000000	Inactivity threshold.
0x26	38	TIME_INACT	R/W	00000000	Inactivity time.
0x27	39	ACT_INACT_CTL	R/W	00000000	Axis enable control for activity and inactivity detection.
0x28	40	THRESH_FF	R/W	00000000	Free-fall threshold.
0x29	41	TIME_FF	R/W	00000000	Free-fall time.
0x2A	42	TAP_AXES	R/W	00000000	Axis control for tap/double tap.
0x2B	43	ACT_TAP_STATUS	R	00000000	Source of tap/double tap.
0x2C	44	BW_RATE	R/W	00001010	Data rate and power mode control.
0x2D	45	POWER_CTL	R/W	00000000	Power-saving features control.
0x2E	46	INT_ENABLE	R/W	00000000	Interrupt enable control.
0x2F	47	INT_MAP	R/W	00000000	Interrupt mapping control.
0x30	48	INT_SOURCE	R	00000010	Source of interrupts.
0x31	49	DATA_FORMAT	R/W	00000000	Data format control.
0x32	50	DATAX0	R	00000000	X-Axis Data 0.
0x33	51	DATAX1	R	00000000	X-Axis Data 1.
0x34	52	DATAY0	R	00000000	Y-Axis Data 0.
0x35	53	DATAY1	R	00000000	Y-Axis Data 1.
0x36	54	DATAZ0	R	00000000	Z-Axis Data 0.
0x37	55	DATAZ1	R	00000000	Z-Axis Data 1.
0x38	56	FIFO_CTL	R/W	00000000	FIFO control.
0x39	57	FIFO_STATUS	R	00000000	FIFO status.

Register 0x00—DEVID (Read Only)

D7	D6	D5	D4	D3	D2	D1	D0
1	1	1	0	0	1	0	1

The DEVID register holds a fixed device ID code of 0xE5
(345 octal).

- DEVID, cihazın yolunda olup olmadığını söyler. DEVID kaydı 0xE5 olarak tutulur yani 0x00 adresi bize 0xE5

olarak dönerse cihaz kullanıma hazır demektir.

- Bunun için okuma fonksiyonu yazıyoruz. Fonksiyonda tanımlamalarımız okuyacağımız register ve kaç byte veri okuyacağımızı yazıyoruz.

```
55 /* USER CODE BEGIN PFP */
56 void Scan_I2C_Adress();
57
58 void ADXL345_Read(uint8_t rRegister, uint8_t numberOfBytes);
59 void ADXL345_Init();
60 /* USER CODE END PFP */
```

```
95 /* USER CODE BEGIN 2 */
96 Scan_I2C_Adress();
97 ADXL345_Init();
98 /* USER CODE END 2 */
```

- ADXL345_Init içerisinde ADXL345_Read fonksiyonu kullanarak 0x00 adresinden 1 byte'lık veriyi okuyacağımı söyleyorum.
- ADXL345_Read fonksiyonunda okuma yapabilmek için yani Master cihaza bağlı Slave cihazın ilgili adresini okuyabilmek için HAL_I2C_Mem_Read fonksiyonunu kullanıyorum.

```
221 void ADXL345_Read(uint8_t rRegister, uint8_t numberOfBytes)
222 {
223     HAL_I2C_Mem_Read(&hi2c1, ADXL345_I2C_SLAVE_ADDRESS, rRegister, 1, myDatas, 1, 100);
224 }
```

```
225 void ADXL345_Init()
226 {
227     ADXL345_Read(0x00, 1);
228 }
```

```
2607 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihazda okuma yapacağım adres oluyor. Biz buraya 0x00 yazmak yerine fonksiyonda yazdığımız rRegister yazıyoruz.
- MemAddSize, okuma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı yazacağımız yeri yazıyoruz. Biz myDatas yazıyoruz.
- Ardından Size kısmına 6 elemanlı dizi olarak tanımladığımızdan 6 yazıyoruz.

```
45 /* USER CODE BEGIN PV */
46 uint8_t i;
47 uint8_t myDatas[6];
48 /* USER CODE END PV */
```

- Size, gönderilecek veri miktarı yazıyoruz.
- 223.satır ve 224.satır break point koyuyoruz ve debug tıklıyoruz. Ardından myDatas değişkenimizi Watch Expression kısmına ekliyoruz. Tarama işlemi bitirdiğinde bize 229 sayısını gösteriyor yani E5 adresi olduğunu gösterir. Böylece ilk kullanıma hazır olduğu anlamına gelir.

Register 0x2D—POWER_CTL (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
0	0	Link	AUTO_SLEEP	Measure	Sleep	Wakeup	

- Her sensörün control registeri vardır. Bizim bunu önce default olarak sıfırlız sonra kendimize göre işlem yaparız.
- Öncelikle Power_CTL registerini sıfırlıyoruz. Bunun için yazma yapacağımızdan ADXL345_Write fonksiyonu tanımlıyoruz.

```
54 /* USER CODE BEGIN PFP */
55 void Scan_I2C_Adress();
56
57 void ADXL345_Write(uint8_t rRegister, uint8_t value);
58 void ADXL345_Read(uint8_t rRegister, uint8_t numberOfBytes);
59 void ADXL345_Init();
60 /* USER CODE END PFP */
```

```

222 void ADXL345_Write(uint8_t rRegister, uint8_t value)
223 {
224     uint8_t data[2] = {0};
225     data[0] = rRegister;
226     data[1] = value;
227
228     HAL_I2C_Master_Transmit(&hi2c1, ADXL345_I2C_SLAVE_ADDRESS, data, 2, 100);
229 }

```

- HAL_I2C_Master_Transmit() fonksiyonuyla yazdırma işlemi yapıyoruz. pData kısmına datayı yazacağımız yer olan data dizisini yazıyoruz. Size olarak dizi iki elemanlı olduğundan 2 yazıyoruz.

```
1032 HAL_StatusTypeDef HAL_I2C_Master_Transmit(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

- Bu şekilde Power_CTL registeri sıfırlanmış oldu.

```

236 void ADXL345_Init()
237 {
238     ADXL345_Read(0x00, 1);
239     ADXL345_Write(0x2D, 0);
240 }

```

- Şimdi Power_CTL registerine işlem yapmak için hangi bitleri set ya da reset yapacağımıza karar veriyoruz.
- Links Bit ile AUTO_SLEEP Bit 0 yapıyoruz.
- Measure Bit, 0 olduğunda bekleme moduna geçerken 1 olduğunda okuma moduna geçiyor. Biz bu biti 1 yapıyoruz.
- Sleep Bit, 0 olduğunda normal mod durumuna getirirken 1 olduğunda uykuya moduna geçer. Biz bu biti 0 yapıyoruz.
- Wakeup Bit, uykuya modunda okuma sıklığını ifade eder. Tabloya göre işlem yaparız. Biz uykuya modunda çalışmadığımızdan D1 ve D0 bitini 0 yapıyoruz.

Setting		Frequency (Hz)
D1	D0	
0	0	8
0	1	4
1	0	2
1	1	1

```

236 void ADXL345_Init()
237 {
238     ADXL345_Read(0x00, 1);
239     ADXL345_Write(0x2D, 0);
240     ADXL345_Write(0x2D, 0x08);      //Measure Bit SET
241 }

```

- g değerini ayarlıyoruz.

Register 0x31—DATA_FORMAT (Read/Write)

D7	D6	D5	D4	D3	D2	D1	D0
SELF_TEST	SPI	INT_INVERT	0	FULL_RES	Justify	Range	

- Data_Format registerinde Range kısmını ayarlıyoruz. +4g için D1 bitini 0, D0 bitini 1 yapıyoruz.

Setting		g Range
D1	D0	
0	0	$\pm 2\text{g}$
0	1	$\pm 4\text{g}$
1	0	$\pm 8\text{g}$
1	1	$\pm 16\text{g}$

```

236 void ADXL345_Init()
237 {
238     ADXL345_Read(0x00, 1);
239     ADXL345_Write(0x2D, 0);
240     ADXL345_Write(0x2D, 0x08);      //Measure Bit SET
241     ADXL345_Write(0x31, 0x01);      //Range Bit +4g
242 }

```

- Veriler 0x32 ile 0x37 registerları arasında toplamda 6 adet saklanır. Bunlar her biri 8 bit olmak üzere DATA X0, DATA X1, DATA Y0, DATA Y1, DATA Z0, DATA Z1'dir.
- 16 bitlik ekseni okumasını ikiye ayırıyor. O yüzden her eksenden iki çıktı veriyor. X ekseni için DATA X0 en düşük 8 bitlik iken DATA X1 en yüksek 8 bittir.

Register 0x32 to Register 0x37—DATA X0, DATA X1, DATA Y0, DATA Y1, DATA Z0, DATA Z1 (Read Only)

- x, y ve z verilerini 16 bit değişkene atayıp birleştireceğiz.

```

45 /* USER CODE BEGIN PV */
46 uint8_t i;
47 uint8_t myDatas[6];
48 int16_t x,y,z;
49 /* USER CODE END PV */

```

- 0x32 adresinden 6 adet byte verisini okumaya başlıyorum.
- Bunlar myDatas dizisinin 0. ve 1. olan byte x; 2. ve 3. byte y; 4. ve 5. byte z verisini temsil eder.
- x, y ve z verilerin en yüksek 8 biti yazarken 0.bitten 8 bit kaydırarak yazıyoruz. Böylece 8 bit ayrılmış iki veriyi 16 bitte birleştirmiş olduk.

```

108 /* USER CODE BEGIN 3 */
109     ADXL345_Read(0x32, 6);
110
111     x= (myDatas[1] << 8) | myDatas[0];
112     y= (myDatas[3] << 8) | myDatas[2];
113     z= (myDatas[5] << 8) | myDatas[4];
114 }
115 /* USER CODE END 3 */

```

- 6 byte veri okuyacağımdan Size kısmı önceden kalan 1 yerine numberOfBytes olarak değiştiriyorum. Buraya 6'da yazabilirdik fakat döngü içerisinde zaten yazdık.

```

237 void ADXL345_Read(uint8_t rRegister, uint8_t numberOfBytes)
238 {
239     HAL_I2C_Mem_Read(&hi2c1, ADXL345_I2C_SLAVE_ADDRESS, rRegister, 1, myDatas, numberOfBytes, 100);
240 }

```

- Her g range'in katsayı değeri vardır.
- Bizim +-4g için 7.8mg/LSB katsayısını alıyoruz. Bunu mg'dan g'ye dönüştürürsek 0.0078 olur.

OUTPUT RESOLUTION	Each axis				
All g Ranges	10-bit resolution	10			Bits
±2 g Range	Full resolution	10			Bits
±4 g Range	Full resolution	11			Bits
±8 g Range	Full resolution	12			Bits
±16 g Range	Full resolution	13			Bits
SENSITIVITY	Each axis				
Sensitivity at Xout, Yout, Zout	±2 g, 10-bit or full resolution	232	256	286	LSB/g
Scale Factor at Xout, Yout, Zout	±2 g, 10-bit or full resolution	3.5	3.9	4.3	mg/LSB
Sensitivity at Xout, Yout, Zout	±4 g, 10-bit resolution	116	128	143	LSB/g
Scale Factor at Xout, Yout, Zout	±4 g, 10-bit resolution	7.0	7.8	8.6	mg/LSB
Sensitivity at Xout, Yout, Zout	±8 g, 10-bit resolution	58	64	71	LSB/g
Scale Factor at Xout, Yout, Zout	±8 g, 10-bit resolution	14.0	15.6	17.2	mg/LSB
Sensitivity at Xout, Yout, Zout	±16 g, 10-bit resolution	29	32	36	LSB/g
Scale Factor at Xout, Yout, Zout	±16 g, 10-bit resolution	28.1	31.2	34.3	mg/LSB
Sensitivity Change Due to Temperature			±0.01		%/°C

```

49 float xg, yg, zg;

```

```

xg = x * .0078;
yg = y * .0078;
zg = z * .0078;

```

Variable Name	Address/Expression	Read Value
xg	0x20000098	0.0312
yg	0x20000094	-0.1014
zg	0x20000088	0.9984

14_02 MPU6050

9 Ocak 2022 Pazar 19:21

14_02 MPU6050

- <https://www.youtube.com/playlist?list=PLmkV2uU8kdplvFZhuZjFI4hEHXdvxzTDw> linkinden takip ediyoruz.
- <https://controllerstech.com/how-to-interface-mpu6050-gy-521-with-stm32/> ve <https://youtu.be/3xSWTsK8YNY> adresinden örnek uygulamayı inceleyebiliriz.
- <https://cbsakademi.ibb.istanbul imu-nedir/> linkinden IMU hakkında bilgi edinebiliriz.
- Kullanacağımız sensör 3 eksen jiroskop ve 3 eksen ivmeölçer ölçümü yapabilmektedir. Bundan sebep 6 eksenli olarak geçiyor. 9 eksenli olanlarda 6 eksenli olanlardan ekstra 3 eksenli manyometre ölçümü yapmaktadır.

➤ İvmeölçer ve Jiroskop Ölçümü

Konfigürasyon Kısımlı

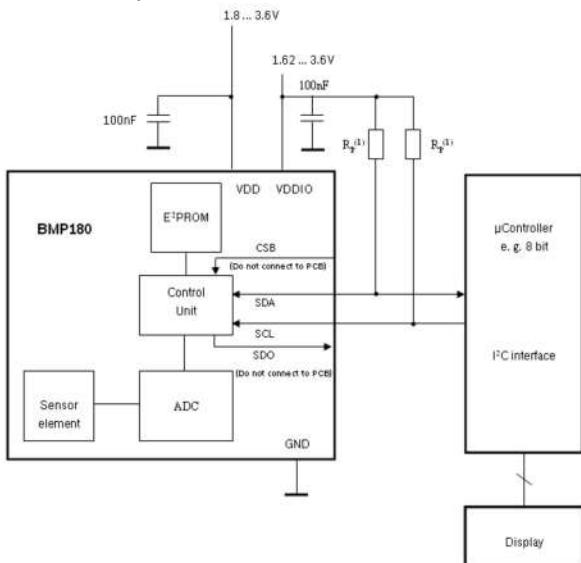
Kod Kısımlı

14_03 BMP180

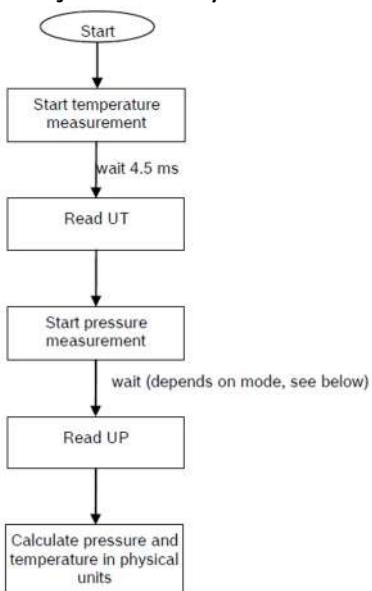
9 Ocak 2022 Pazar 19:21

14_03 BMP180

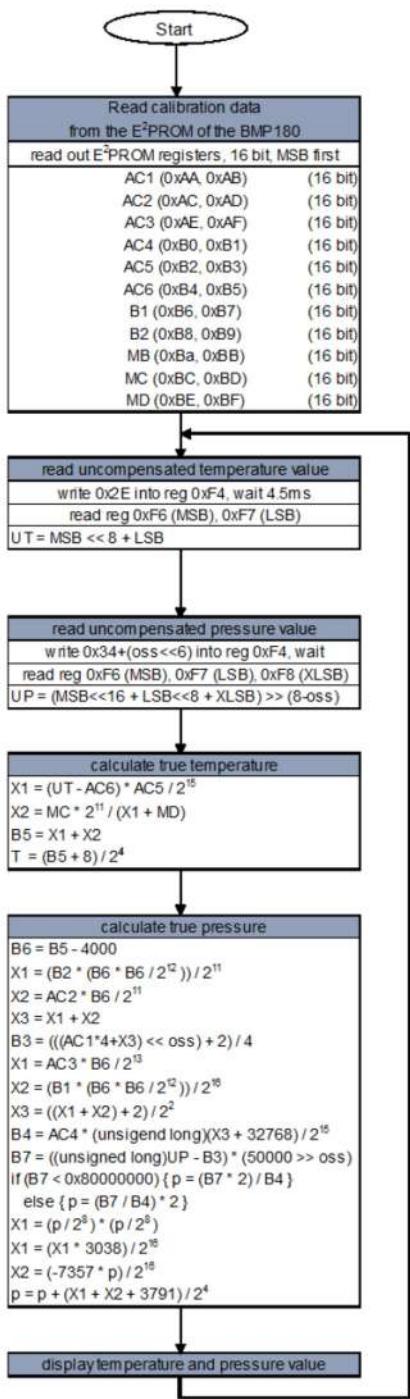
- <https://controllerstech.com/interface-bmp180-with-stm32/> adresinden örnek uygulamayı inceleyebiliriz.



- BMP180 sensörü dengelenmemiş sıcaklık ve basınç verilerini sunar.
- E2PROM 176 bit kalibrasyon datasını saklar. Bu değerler ile sıcaklık ve basınç değerlerini doğrularız.
- UP, pressure datadır. 16bit ile 19bittir. UT, temperature datadır. 16bittir.
- Mikrodenetleyici, sıcaklık ve basınç ölçümünü başlatmak için bir başlatma sinyalini gönderir. Daha sonra I2C üzerinden sıcaklık ve basınç verisi okunur. Sıcaklığı °C ve basıncı hPa cinsinden okumak için kalibrasyon verileri okunmalıdır. Bunun için I2C aracılığıyla E2PROM'dan bakılır.

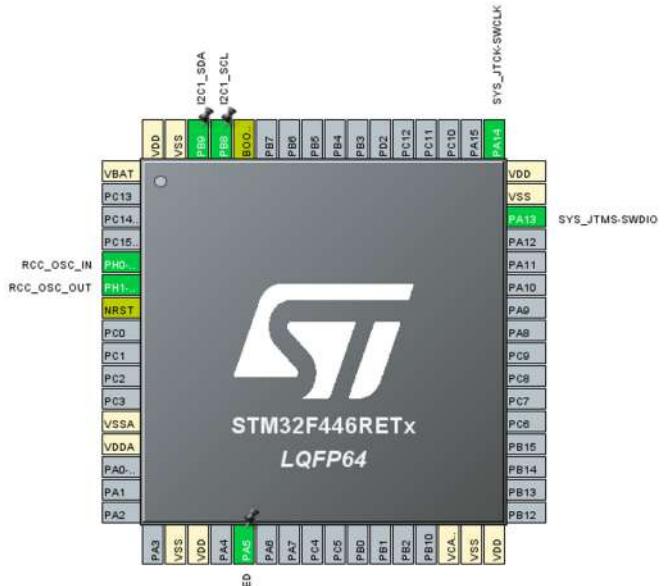


- Aşağıda diyagram üzerinden sensörden nasıl sıcaklık ve basınç değerinin okunduğunun işleyişi anlatılıyor.



example:	C code function:	type:
	bmp180_get_cal_param	
AC1 = 408 AC2 = -72 AC3 = -14383 AC4 = 32741 AC5 = 32757 AC6 = 23153 B1 = 6190 B2 = 4 MB = -32768 MC = -8711 MD = 2868		short short short unsigned short unsigned short unsigned short short short short short short short
	bmp180_get_ut	
UT = 27898		long
oss = 0 = oversampling_setting (ultra low power mode)	bmp180_get_up	short (0 .. 3)
UP = 23843		long
	bmp180_get_temperature	
X1 = 4743 X2 = -2344 B5 = 2399 T = 150		long long long temp in 0.1°C
	BMP180_calpressure	
B6 = -1601 X1 = 1 X2 = 56 X3 = 57 B3 = 422 X1 = 2810 X2 = 59 X3 = 717 B4 = 33457 B7 = 1171050000 p = 70003		long long long long long long long unsigned long unsigned long long long long long long
X1 = 74529 X1 = 3454 X2 = -7859 p = 69964		long long long long press. in Pa

➤ Sıcaklık ve Basınç Okuma Konfigürasyon Kısımlı



Pin N...	Signal on	GPIO out...	GPIO mo...	GPIO Pull...	Maximum...	User Label	Modified
Pin Name	Signal on Pin	GPIO output	GPIO mode	GPIO Pull-up	Maximum ou...	User Label	Modified
PA5	n/a	Low	Output Pu...	No pull-up	Low	LED	<input checked="" type="checkbox"/>
PB8	I2C1_SCL	n/a	Alternate Fu...	Pull-up	Very High		<input type="checkbox"/>
PB9	I2C1_SDA	n/a	Alternate Fu...	Pull-up	Very High		<input type="checkbox"/>

- Connectivity başlığı altında I2C seçip Mode ekranından I2C seçiyoruz.

I2C I2C

- I2C için Speed Mode olarak Fast Mode seçimi yapıyoruz.

Master Features

I²C Speed Mode Fast Mode

I2C Clock Speed (Hz) 400000

Fast Mode Duty Cycle Duty cycle $T_{low}/T_{high} = 2$

Slave Features

Clock No Stretch Mode Disabled

Primary Address Length s...7-bit

Dual Address Acknowledg. Disabled

Primary slave address 0

General Call address dete.. Disabled

Kod Kısmı

- Dosyamızın Inc klasörü için bmp180.h adında Header File, Src klasörüne bmp180.c adında Source File ekliyoruz.
 - Header File içerisinde fonksiyon tanımları, veri türü tanımları ve makrolar barındırır.

```
8 #ifndef INC_BMP180_H_
9 #define INC_BMP180_H_
10
11
12
13 #endif /* INC_BMP180_H_
```

- Ardından birkaç fonksiyon ekliyorum. Sensörü başlatmak ve kalibrasyon için fonksiyon yazıyoruz.
 - Sensörle haberleşme yaparken okuma ve yazma yapabilmek için kullanacağım adresleri tanımlıyorum.
 - BMP180 adresine okuma yapmak için ilk bit 1 yapılır ve 0xEF adresine karşılık gelir. Yazma yapmak için ilk bit 0 yapılır, 0xEE adresine karşılık gelir.

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

- Böylece yazma için 0xEE, okuma için 0xEF adreslerini tanımlıyorum.

```
11 #define BMP180_DEVICE_WRITE_REGISTER_ADDRESS 0xEE  
12 #define BMP180_DEVICE_READ_REGISTER_ADDRESS 0xEF  
13  
14 void BMP180_Init();  
15 void BMP180_Get_Calibration_Value();
```

- Bunları yazdıktan sonra Init fonksiyonun içeriğini yazmaya başlıyoruz.

- Her çevre birimin control registeri vardır. Biz datasheet'ten Memory Map tablosunu inceliyoruz.

Register Name	Register Adress	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
out_xlsb	F8h			adc_out_xlsb<7:3>			0	0	0	00h
out_lsb	F7h				adc_out_lsb<7:0>					00h
out_msb	F6h				adc_out_msb<7:0>					80h
ctrl_meas	F4h		oss<1:0>		sco				measurement control	00h
soft reset	E0h					reset				00h
id	D0h					id<7:0>				55h
calib21 downto calib0	Bfh down to AAh					calib21<7:0> down to calib0<7:0>				n/a

Registers:	Control registers	Calibration registers	Data registers	Fixed
Type:	read / write	read only	read only	read only

- Ctrl_meas adresi 0xF4'tür. Adresin 8 bitin ilk 4 biti measurement control, 5.bit sco, 6. ve 7.bit oss'dır.
- sco, Start of conversion yani çevrim başlangıcıdır.
- Oss, oversampling_setting için girilen değerdir.
- Basıncı farklı modlar kullanarak optimum güç tüketimi, hız ve çözünürlük arasında seçim yapabiliriz. Bunun için oss değerine girilecek bitler ultra low power için 00, standard 01, high resolution için 10, ultra high resolution 11'dir.

Mode	Parameter oversampling_setting	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [µA]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

- Kütüphanelerde I2C kullanabilmek için main içerisinde tanımlanmış kütüphanemize ekliyoruz. Bunu tanımlayabilmek için öncesinde stm32f4xx_hal.h kütüphanesini dahil ediyoruz.

```
11 #include "stm32f4xx_hal.h"
12
13 extern I2C_HandleTypeDef hi2c1;
```

- Ardından HAL_I2C_IsDeviceReady() fonksiyonunu kullanıyoruz.

```
3420 HAL_StatusTypeDef HAL_I2C_IsDeviceReady(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint32_t Trials, uint32_t Timeout)
```

- hal_i2c.c dosyasından ulaştığımız HAL_I2C_IsDeviceReady() fonksiyonu ile hedef cihazın iletişim için hazır olup olmadığını kontrol ederiz.
- İf yapısı içerisinde 1 değil ise ledi yakar yani sensörün işlemci ile bağlantısı yoksa led yanar.

```
10 void BMP180_Init()
11 {
12     if(HAL_I2C_IsDeviceReady(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 1, 100000) != HAL_OK)
13         HAL_GPIO_WritePin(GPIOD, GPIO_PIN_5, GPIO_PIN_SET);
14 }
```

- Bu uygulamayı görebilmek için main.c dosyasına bmp.h kütüphanesini ekleriz ve BMP180_Init() fonksiyonunu çağırırız.

```
23 /* Private includes */
24 /* USER CODE BEGIN Includes */
25 #include "bmp180.h"
26 /* USER CODE END Includes */
```

```
90 /* USER CODE BEGIN 2 */
91 BMP180_Init();
92 /* USER CODE END 2 */
```

- Asıl initialization işlemi yani başlatma işlemi kalibrasyon işlemi için E2PROM'dan çekilecek verilerdir.

- 176 bitlik E2PROM her biri 16 bitlik 11 kelimeye bölünmüştür. Her sensör modülün ayrı katsayıları vardır.
- Sıcaklık ve basınç hesabı yapmadan önce E2PROM verileri okunur. Veri haberleşmesinde her bir kelimenin 0 ya da 0xFFFF değeri olmadığı kontrol edilir.

BMP180 reg adr		
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

- Bu verileri bmp180.c dosyasında tanımlıyoruz.

```

18 short AC1 = 0;
19 short AC2 = 0;
20 short AC3 = 0;
21 unsigned short AC4 = 0;
22 unsigned short AC5 = 0;
23 unsigned short AC6 = 0;
24 short B1 = 0;
25 short B2 = 0;
26 short MB = 0;
27 short MC = 0;
28 short MD = 0;

```

- Bunlar için kaç adet olduğuna dair tanımlama yapıyorum. Her biri 16 bit olduğunu biliyorum. Tanımlarken 8 bit olarak tanımlama yapacağımından toplam veri 22 adet oluyor.

```
20 #define BMP180_CALIBRATION_VALUE_LENGTH 22
```

- bmp180.h dosyasında void BMP180_Get_Calibration_Value() fonksiyonu tanımlıyorum ve bmp.c dosyasında fonksiyonu yazıyoruz. Önce 22 veriyi dizİYE atıp değerlerini sıfırlıyoruz.

```
uint8_t Calib_Buff[BMP180_CALIBRATION_VALUE_LENGTH] = {0};
```

- Okuma işlemini HAL_I2C_Mem_Read() fonksiyonu ile yapıyorum.

```
2687 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihazda okuma yapacağım adres oluyor. Biz buraya verileri okuyacağımız ilk veri olan AC1'in başlangıç adresi olan 0xAA yazıyoruz.

Bunun içinde bmp180.h dosyasında tanımlama yapıyorum.

```
23 #define BMP180_CALIBRATION_ADDRESS_START 0XAA
```

- MemAddSize, okuma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı yazacağımız yeri yazıyoruz. 22 veriyi tanımladığımız dizinin ismini yazıyoruz.
- Ardından Size kısmına 22 elemanlı dizi olarak tanımladığımızdan 22 yazıyoruz.

```
HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, BMP180_CALIBRATION_ADDRESS_START, 1, Calib_Buff, BMP180_CALIBRATION_VALUE_LENGTH, 100);
```

- Okuma işlemini yapıp bunları dizİYE kaydettik. Bu kaydedilen verileri tanımladığımız verilere yazıyoruz. Biz dizİYE kaydederken 8 bit olarak kaydediyorduk ama verimiz 16 bit olduğundan ilk 8 biti 8 bit sola kaydediyoruz. İkinci 8 biti aynen yazıyoruz.

```

AC1 = ((Calib_Buff[0] << 8) | Calib_Buff[1]);
AC2 = ((Calib_Buff[2] << 8) | Calib_Buff[3]);
AC3 = ((Calib_Buff[4] << 8) | Calib_Buff[5]);
AC4 = ((Calib_Buff[6] << 8) | Calib_Buff[7]);
AC5 = ((Calib_Buff[8] << 8) | Calib_Buff[9]);
AC6 = ((Calib_Buff[10] << 8) | Calib_Buff[11]);
B1 = ((Calib_Buff[12] << 8) | Calib_Buff[13]);
B2 = ((Calib_Buff[14] << 8) | Calib_Buff[15]);
MB = ((Calib_Buff[16] << 8) | Calib_Buff[17]);
MC = ((Calib_Buff[18] << 8) | Calib_Buff[19]);
MD = ((Calib_Buff[20] << 8) | Calib_Buff[21]);

```

```

31 void BMP180_Get_Calibration_Value()
32 {
33     uint8_t Calib_Buff[BMP180_CALIBRATION_VALUE_LENGTH] = {0};
34
35     HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, BMP180_CALIBRATION_ADDRESS_START, 1, Calib_Buff, BMP180_CALIBRATION_VALUE_LENGTH, 100);
36
37     AC1 = ((Calib_Buff[0] << 8) | Calib_Buff[1]);
38     AC2 = ((Calib_Buff[2] << 8) | Calib_Buff[3]);
39     AC3 = ((Calib_Buff[4] << 8) | Calib_Buff[5]);
40     AC4 = ((Calib_Buff[6] << 8) | Calib_Buff[7]);
41     AC5 = ((Calib_Buff[8] << 8) | Calib_Buff[9]);
42     AC6 = ((Calib_Buff[10] << 8) | Calib_Buff[11]);
43     B1 = ((Calib_Buff[12] << 8) | Calib_Buff[13]);
44     B2 = ((Calib_Buff[14] << 8) | Calib_Buff[15]);
45     MB = ((Calib_Buff[16] << 8) | Calib_Buff[17]);
46     MC = ((Calib_Buff[18] << 8) | Calib_Buff[19]);
47     MD = ((Calib_Buff[20] << 8) | Calib_Buff[21]);
48 }

```

- Yazdığımız BMP180_Get_Calibration_Value() fonksiyonunu BMP180_Init() fonksiyonu içerisinde ekleriz.

```

23 void BMP180_Init()
24 {
25     if(HAL_I2C_IsDeviceReady(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 1, 100000) != HAL_OK)
26         HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
27
28     BMP180_Get_Calibration_Value();
29 }

```

- STMStudio programını açıp File kısmından Import variables tıklıyoruz ve çalışma dosyamızın debug içerisindeki elf. Uzantılı dosyayı seçiyoruz ve içerisinde okuyacağımız değişkeni import edip yandaki grafiğe ekliyoruz.
- Sonuçlar sabit değerlerdir.

Variable Name	Address/Expression	Read Value
AC1	0x20000028	7587
AC2	0x2000002a	-998
AC3	0x2000002c	-14362
AC4	0x2000002e	32452
AC5	0x20000030	25627
AC6	0x20000032	14728
B1	0x20000034	6515
B2	0x20000036	32
MB	0x20000038	-32768
MC	0x2000003a	-11786
MD	0x2000003c	2438

- Böylelikle sıcaklık ve basınç değerlerini hesaplayabiliriz.
- Void yazmamızın sebebi fonksiyonlar bir parametre istemiyor ve geri döndürülmesine gerek olmadığından yazıyoruz ama sıcaklık ve basınç değerlerini hesapladığımız fonksiyonları geri değer döndürüyoruz.

```

25 void BMP180_Init(void);
26 void BMP180_Get_Calibration_Value(void);
27 void BMP180_Get_Uncompensated_Temperature(void);
28 float BMP180_Get_Temperature(void);
29 void BMP180_Get_Uncompensated_Pressure(void);
30 float BMP180_Get_Pressure(void);

```

- Ardından hesaplama için gerekli değişkenleri yazıyorum.

```

24 //Calculate Value
25 long UT = 0;
26 short oss = 3;           //over sampling setting 0,1,2,3
27 long UP = 0;
28 long X1 = 0;
29 long X2 = 0;
30 long X3 = 0;
31 long B3 = 0;
32 long B5 = 0;
33 unsigned long B4 = 0;
34 long B6 = 0;
35 unsigned long B7 = 0;
36
37 long Temp = 0;
38 long Press = 0;

```

- Sıcaklık değerini hesaplamak için önce BME180_Get_Uncompensated_Temperature() fonksiyonunu yazıyoruz. Bunun için datasheet kısmına baktığımızda yapılacak ilk işlem 0xF4 adresine 0x2E datasını yazıyoruz. Ardından 4.5ms bekliyoruz. Daha sonra 0xF6 ve 0xF7 adreslerinden okuma yapıyorum. Bu adreslerde okuduğumuz 8 bitlik 2 datayı 16 bitlik yeni değere yazıyoruz.
- Yazma işlemini HAL_I2C_Mem_Write() fonksiyonu ile yapıyorum.

```
2484 HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihaza yazma yapacağım adres oluyor. Biz buraya 0xF4 yazıyoruz.
- MemAddSize, yazma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı okuyacağımız yeri yazıyoruz. Bunun için fonksiyon içinde 0x2E adresini tanımlıyorum. Bu şekilde tanımladığımızdan fonksiyon içinde pointer için başına & işaretü koyuyorum.

```
uint8_t datatowrite = 0x2E;
```

- Ardından Size kısmına 1 yazıyoruz.

```
HAL_I2C_Mem_Write(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 0xF4, 1, &datatowrite, 1, 1000);
```

- Bekleme için delay yazıyoruz.

```
HAL_Delay(5);
```

- Okuma işlemini HAL_I2C_Mem_Read() fonksiyonu ile yapıyorum.

```
2607 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihazda okuma yapacağım adres oluyor. Biz buraya verileri okuyacağımız ilk veri olan 0xF6 yazıyoruz.
- MemAddSize, okuma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı yazacağımız yeri yazıyoruz. 2 veriyi tanımladığımız dizinin ismini yazıyoruz.

```
uint8_t datatoread[2] = {0};
```

- Ardından Size kısmına 2 elemanlı dizi olarak tanımladığımızdan 2 yazıyoruz.

```
HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, 0xF6, 1, datatoread, 2, 1000);
```

- 0xF6 adresinden okuduğum 8 bitlik datayı 8 bit sola kaydırıyorum ardından 0xF7 adresindeki 8 bitlik datayı direk yazıyorum.

```
UT = ((datatoread[0] << 8) | datatoread[1]);
```

```

67 void BMP180_Get_Uncompensated_Temperature(void)
68 {
69     uint8_t datatowrite = 0x2E;
70     HAL_I2C_Mem_Write(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 0xF4, 1, &datatowrite, 1, 1000);
71     HAL_Delay(5);
72
73     uint8_t datatoread[2] = {0};
74     HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, 0xF6, 1, datatoread, 2, 1000);
75     UT = ((datatoread[0] << 8) | datatoread[1]);
76 }

```

- UT değerini görebilmek için yazdığımız BMP180_Get_Uncompensated_Temperature() fonksiyonunu main.c dosyasına ekliyoruz.

```

91 /* USER CODE BEGIN 2 */
92 BMP180_Init();
93 BMP180_Get_Uncompensated_Temperature();
94 /* USER CODE END 2 */

```

Variable Name	Address/Expression	Read Value
UT	0x20000040	23310

- Asıl şimdilik sıcaklık değerini bulacağız.
- Temp değeri 0.1°C 'deki değeri olduğundan 10 ile bölerz ya da 0.1 ile çarparız.

```

78 float BMP180_Get_Temperature(void)
79 {
80     BMP180_Get_Uncompensated_Temperature();
81
82     X1 = ((UT-AC6) * (AC5/(pow(2,15))));;
83     X2 = ((MC*(pow(2,11))) / (X1+MD));
84
85     B5 = X1+X2;
86     Temp = (B5+8)/(pow(2,4));
87     return Temp*0.1;
88 }

45 /* USER CODE BEGIN PV */
46 float temperature=0;
47 /* USER CODE END PV */

96 /* Infinite loop */
97 /* USER CODE BEGIN WHILE */
98 while (1)
99 {
100     /* USER CODE END WHILE */
101
102     /* USER CODE BEGIN 3 */
103     temperature = BMP180_Get_Temperature();
104 }
105 /* USER CODE END 3 */
106 }

```

Variable Name	Address/Expression	Read Value
temperature	0x20000054	25.5

- Basınç değerini hesaplamak için önce BME180_Get_Uncompensated_Pressure() fonksiyonunu yazıyoruz. Bunun için datasheet kısmına baktığımızda yapılacak ilk işlem 0xF4 adresine 0x34 datasını ve buna oss değerini sola 6 bit kaydırarak ekleyerek yazıyoruz. Ardından oss değerine göre datasheet'deki Conversion Time için verilen ms değeri kadar bekleriz. Daha sonra 0xF6, 0xF7 ve 0xF8 adreslerinden okuma yapıyorum. Bu adreslerden okuduğumuz 8 bitlik 3 datayı 24 bitlik yazıyorum ve 8'den oss değerini çıkarıp ardından bu sayıyı 24 bitlik değer kadar sağa kaydırma yapıyorum.
- Oss için ultra high resolution yani ultra yüksek çözünürlük seçimi yapıyorum. Bunun parametre değeri 3'tür.

Fonksiyon içinde tanımlıyorum.

```
short oss = 3;
```

- Yazma işlemini HAL_I2C_Mem_Write() fonksiyonu ile yapıyorum.

```
2484 HAL_StatusTypeDef HAL_I2C_Mem_Write(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout);
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihaza yazma yapacağım adres oluyor. Biz buraya 0xF4 yazıyoruz.
- MemAddSize, yazma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı okuyacağımız yeri yazıyoruz. Bunun için fonksiyon içinde datatowrite değişkenine yazdığını adresini tanımlıyorum. Bu şekilde tanımladığımızdan fonksiyon içinde pointer için başına & işaretini koyuyorum.

```
uint8_t datatowrite = 0x34 | (0x03 << 6);
```

- Ardından Size kısmına 1 yazıyoruz

```
HAL_I2C_Mem_Write(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 0xF4, 1, &datatowrite, 1, 1000);
```

- Bekleme için delay yazıyoruz.

```
HAL_Delay(5);
```

- Okuma işlemini HAL_I2C_Mem_Read() fonksiyonu ile yapıyoruz.

```
2607 HAL_StatusTypeDef HAL_I2C_Mem_Read(I2C_HandleTypeDef *hi2c, uint16_t DevAddress, uint16_t MemAddress, uint16_t MemAddSize, uint8_t *pData, uint16_t Size, uint32_t Timeout)
```

- DevAddress, Slave cihaz için tanımladığımız adres oluyor.
- MemAddress, Slave cihazda okuma yapacağım adres oluyor. Biz buraya verileri okuyacağımız ilk veri olan 0xF6 yazıyoruz.
- MemAddSize, okuma yapılacak adresin boyutunu yazıyoruz. 8 bit veri okuması yapacağımızdan 1 yazıyoruz.
- pData, datayı yazacağımız yeri yazıyoruz. 3 veriyi tanımladığımız dizinin ismini yazıyoruz.

```
uint8_t datatoread[3] = {0};
```

- Ardından Size kısmına 3 elemanlı dizi olarak tanımladığımızdan 3 yazıyoruz.

```
HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, 0xF6, 1, datatoread, 3, 1000);
```

- 0xF6 adresinden okuduğum 8 bitlik datayı 16 bit sola kaydırıyorum ardından 0xF7 adresindeki okuduğum 8 bitlik datayı 8 bit sola kaydırıyorum ardından 0xF8 adresindeki 8 bitlik datayı direk yazıyorum.

```
UP = ((datatoread[0] << 16) | (datatoread[1] << 8) | datatoread[2]) >> (8-oss));
```

```
90 void BMP180_Get_Uncompensated_Pressure(void)
91 {
92     short oss = 3;
93     uint8_t datatowrite = 0x34 | (oss << 6);
94     HAL_I2C_Mem_Write(&hi2c1, BMP180_DEVICE_WRITE_REGISTER_ADDRESS, 0xF4, 1, &datatowrite, 1, 1000);
95     switch (oss)
96     {
97         case (0):
98             HAL_Delay (5);
99             break;
100        case (1):
101            HAL_Delay (8);
102            break;
103        case (2):
104            HAL_Delay (14);
105            break;
106        case (3):
107            HAL_Delay (26);
108            break;
109    }
110    uint8_t datatoread[3] = {0};
111    HAL_I2C_Mem_Read(&hi2c1, BMP180_DEVICE_READ_REGISTER_ADDRESS, 0xF6, 1, datatoread, 3, 1000);
112    UP = ((datatoread[0] << 16) | (datatoread[1] << 8) | datatoread[2]) >> (8-oss));
113 }
```

- UP değerini görebilmek için yazdığımız BMP180_Get_Uncompensated_Pressure() fonksiyonunu main.c dosyasına ekliyoruz.

```
92 /* USER CODE BEGIN 2 */
93 BMP180_Init();
94 BMP180_Get_Uncompensated_Temperature();
95 BMP180_Get_Uncompensated_Pressure();
96 /* USER CODE END 2 */
```

Variable Name	Address/Expression	Read Value
UP	0x20000044	322467

- Asıl şimdi basınç değerini bulacağız.
- Press değeri Pa cinsindendir.

```
115 float BMP180_Get_Pressure(void)
116 {
117     BMP180_Get_Uncompensated_Pressure();
118
119     short oss = 3;
120     B6 = B5-4000;
121     X1 = (B2 * (B6*B6/(pow(2,12))))/(pow(2,11));
122     X2 = AC2*B6/(pow(2,11));
123     X3 = X1+X2;
124     B3 = (((AC1*X3)+X3)<oss)+2)/4;
125     X1 = AC3*B6/pow(2,13);
126     X2 = (B1 * (B6*B6/(pow(2,12))))/(pow(2,16));
127     X3 = ((X1+X2)+2)/pow(2,2);
128     B4 = AC4*(unsigned long)(X3+32768)/(pow(2,15));
```

```

115 float BMP180_Get_Pressure(void)
116 {
117     BMP180_Get_Uncompensated_Pressure();
118
119     short oss = 3;
120     B6 = B5-4000;
121     X1 = (B2 * (B6*B6/(pow(2,12))))/(pow(2,11));
122     X2 = AC2*B6/(pow(2,11));
123     X3 = X1+X2;
124     B3 = (((AC1*4+X3)<oss)+2)/4;
125     X1 = AC3*B6/pow(2,13);
126     X2 = (B1 * (B6*B6/(pow(2,12))))/(pow(2,16));
127     X3 = ((X1+X2)+2)/pow(2,2);
128     B4 = AC4*(unsigned long)(X3+32768)/(pow(2,15));
129     B7 = ((unsigned long)UP-B3)*(50000>>oss);
130     if (B7<0x80000000) Press = (B7*2)/B4;
131     else Press = (B7/B4)*2;
132     X1 = (Press/(pow(2,8)))*(Press/(pow(2,8)));
133     X1 = (X1*3038)/(pow(2,16));
134     X2 = (-7357*Press)/(pow(2,16));
135     Press = Press + (X1+X2+3791)/(pow(2,4));
136
137 }
138
139 /* USER CODE BEGIN PV */
140 float temperature=0;
141 float pressure=0;
142 /* USER CODE END PV */
143
144
145 /* Infinite loop */
146 /* USER CODE BEGIN WHILE */
147 while (1)
148 {
149     /* USER CODE END WHILE */
150
151     /* USER CODE BEGIN 3 */
152     temperature = BMP180_Get_Temperature();
153     pressure = BMP180_Get_Pressure();
154 }
155 /* USER CODE END 3 */
156

```

Variable Name	Address/Expression	Read Value
pressure	0x20000074	101006.0

- Artık irtifayı hesaplayabiliriz. İrtifa hesaplamak için gerekli formül aşağıdadır.

$$\text{altitude} = 44330 \times \left(1 - \left(\frac{p}{p_0} \right)^{\frac{1}{5.255}} \right)$$

- P değeri bizim bulduğumuz basınç değeridir.
- P0 değeri yani Standart Atmosferik Basınç değeri için bmp180.h dosyasına 101325Pa değerini tanımlıyoruz.

```

25 //Standard Atmospheric Pressure
26 #define atmPress 101325

```

- Önce formülü yazacağımız bmp180.h dosyasına fonksiyonu tanımlıyoruz ardından bmp180.c dosyasında formülü yazıyoruz.

```
34 float BMP180_Get_Altitude(void);
```

```

139 float BMP180_Get_Altitude(void)
140 {
141     BMP180_Get_Pressure();
142
143     return 44330*(1-(pow((float)Press/(float)atmPress), 0.19029495718));
144 }

45 /* USER CODE BEGIN PV */
46 float temperature=0;
47 float pressure=0;
48 float altitude=0;
49 /* USER CODE END PV */

```

```

99  /* Infinite loop */
100 /* USER CODE BEGIN WHILE */
101 while (1)
102 {
103     /* USER CODE END WHILE */
104
105     /* USER CODE BEGIN 3 */
106     temperature = BMP180_Get_Temperature();
107     pressure = BMP180_Get_Pressure();
108     altitude = BMP180_Get_Altitude();
109 }
110 /* USER CODE END 3 */
111 }
```

- Bulunduğumuz bölgenin irtifası 20m üzerindedir.

Variable Name	Address/Expression	Read Value
altitude	0x200000e0	22.669767

- BMP180 sensöründen okuma yaptığımız basınç, sıcaklık verisi ile basınç verisini kullanarak elde ettiğimiz irtifa verisinin sonuçları aşağıdadır.

Variable Name	Address/Expression	Read Value
altitude	0x200000e0	21.501776
pressure	0x200000dc	101068.0
temperature	0x200000d8	25.5

14_04 OLED

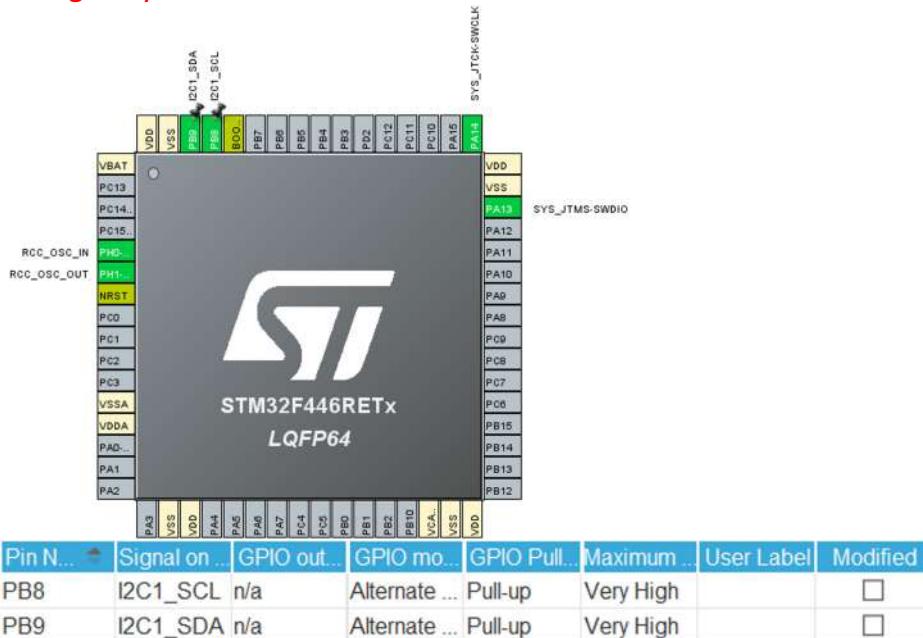
9 Ocak 2022 Pazar 19:21

14_04 OLED

- <https://controllerstech.com/oled-display-using-i2c-stm32/> adresinden örnek uygulamayı inceleyebiliriz.

➤ Ekran Kullanımı

Konfigürasyon Kısmı



- Connectivity başlığı altında I2C seçip Mode ekranından I2C seçiyoruz.

I2C **I2C**

- I2C için Speed Mode olarak Fast Mode seçimi yapıyoruz.

Master Features

I2C Speed Mode	Fast Mode
I2C Clock Speed (Hz)	400000
Fast Mode Duty Cycle	Duty cycle Tlow/Thigh = 2

Slave Features

Clock No Stretch Mode	Disabled
Primary Address Length s...	7-bit
Dual Address Acknowledg...	Disabled
Primary slave address	0
General Call address dete...	Disabled

Kod Kısmı

- Dosyamızın Inc klasörü için fonts.h, test.h ve ssd1306.h adında Header File, Src klasörüne fonts.c, test.c ve ssd1306.c adında Source File ekliyoruz.
- Kod içerisinde eklediğimiz dosyaları ekliyoruz.

```
23 /* Private includes -----  
24 /* USER CODE BEGIN Includes */  
25 #include "ssd1306.h"  
26 #include "fonts.h"  
27 /* USER CODE END Includes */
```

- Eklenen dosyalar içerisinde header dosyalarına stm32f4xx_hal.h ekliyoruz.

```
#include "stm32f4xx_hal.h"
```

- Eklenen dosyalar içerisinde source dosyalarına I2C tanımlaması ekliyoruz.

```
extern I2C_HandleTypeDef hi2c1;
```

```
92 /* USER CODE BEGIN 2 */  
93 SSD1306_Init();  
94  
95 SSD1306_GotoXY (10,10);  
96 SSD1306_Puts ("HELLO", &Font_11x18, 1);  
97 SSD1306_GotoXY (10, 30);  
98 SSD1306_Puts ("WORLD !!", &Font_11x18, 1);  
99 SSD1306_UpdateScreen();  
100  
101 HAL_Delay(500);
```

14_05 NRF24L01

9 Ocak 2022 Pazar 19:22

14_05 NRF24L01

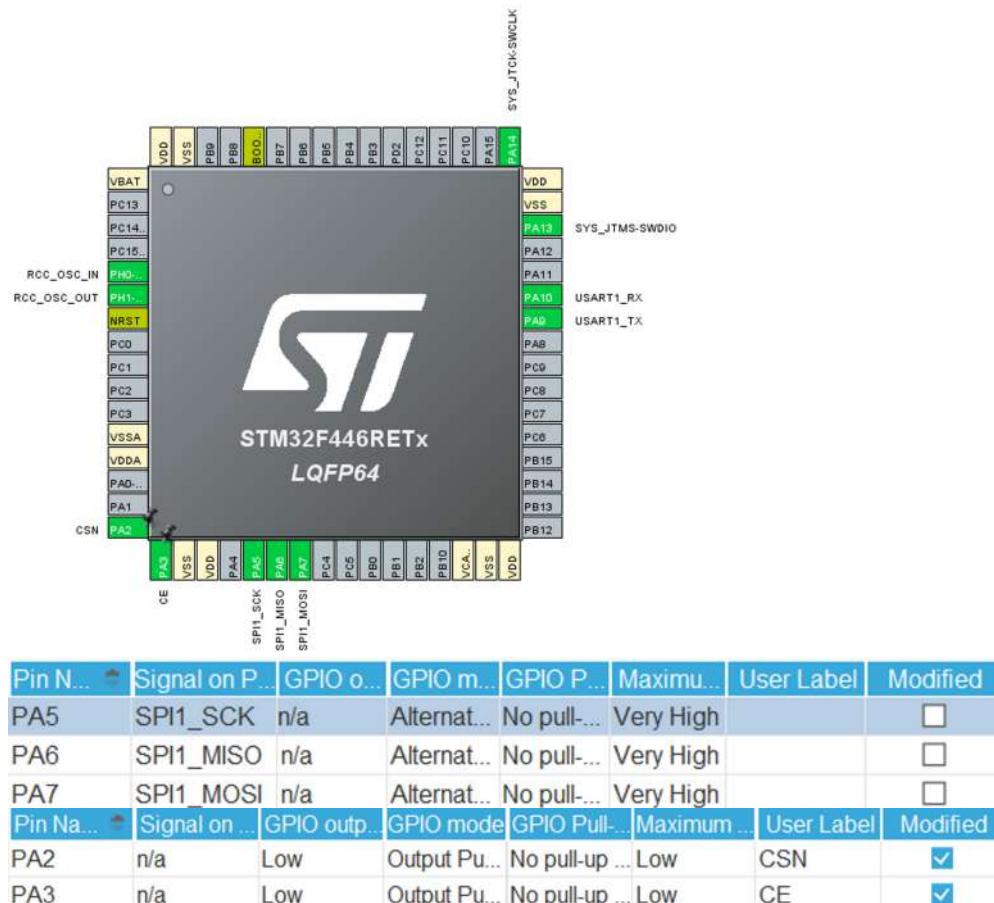
- Örnek için <https://milisaniye.home.blog/2020/09/27/stm32-nrf24l01-wifi-modulu-uygulamasi/> ile <https://youtu.be/O2dg2Eo7vo8> linklerden yararlanıyoruz.
- CE, modülü aktif hale getirirken CSN ise SPI haberleşmeyi aktif ediyor. CE low iken NRF modülün üzerindeki çipin gücünü kapatmış oluyoruz. Konfigürasyon ayarlarında CE ve CSN pinini output olarak kullanıyoruz.
- Modül transmit ve receive işlemini gerçekleştirirken bir interrupt oluşturuyor ve bunu IRQ pini ile gerçekleştiriyor. Konfigürasyon ayarlarında IRQ pini için input değil exti olarak kullanıyoruz.

➤ Giriş

- MY_NRF24.h header dosyasında yüklediğimiz kartın hal kütüphanesini tanımlamalıyız. STM32F103C8T6 için #include "stm32f1xx_hal.h", STM32F446RE için #include "stm32f4xx_hal.h" olarak düzeltiyoruz.

Verici

Konfigürasyon Kısımlı



- Connectivity Başlığı altından SPI1'i seçip Mode ekranından Full-Duplex Master'i seçiyoruz.
- Full-Duplex çift yönlü haberleşme demektir.

Mode Full-Duplex Master

Hardware NSS Signal Disable

- Prescaler değerini 32 yaptığımızda clock ayarımıza göre Baud Rate'i Cubemx kendisi ayarlıyor. Baud rate'i değiştirmek istiyorsanız Prescaler değerini veya clock değerini değiştirmeniz gerekiyor.

Basic Parameters	
Frame Format	Motorola
Data Size	8 Bits
First Bit	MSB First

Clock Parameters	
Prescaler (for Baud Rate)	32
Baud Rate	2.25 MBits/s
Clock Polarity (CPOL)	Low
Clock Phase (CPHA)	1 Edge

Advanced Parameters	
CRC Calculation	Disabled
NSS Signal Type	Software

- CSN ve CE pinleri Output modundadır.
- Verici kısmında USART tam olarak kullanmasak da tanımlama yapmamız gerekiyor.
- Connectivity Başlığı altında USART1'i seçip Mode ekranından Asynchronous'u seçiyoruz. Baud Rate değerininide 115200 yapıyoruz tabi farklı değerde seçebiliriz.

Mode	Asynchronous	▼
Hardware Flow Control (RS232)	Disable	▼

Basic Parameters	
Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

Advanced Parameters	
Data Direction	Receive and Transmit
Over Sampling	16 Samples

- Clock değerini alıcı kart ile aynı olması için 72 Mhz seçti. Her iki kart için farklı bir değer seçebiliriz ama Baud Rate'in aynı olmasına dikkat edilmeli.

Kod Kısmı

- MY_NRF24.h dosyasında düzeltme yapıyoruz.

```
125 #include "stm32f4xx_hal.h"
• MY_NRF24.c dosyasına stdio.h kütüphanesini ekliyoruz.
21 #include "MY_NRF24.h"
22 #include "stdio.h"
• Kütüphanemizi dahil ettik.
```

```
23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #include "MY_NRF24.h"
26 /* USER CODE END Includes */
```

```
57 /* Private user code -----
58 /* USER CODE BEGIN 0 */
59 uint64_t TxpipeAddrs = 122;
60 char myTxData[32] = "Mili Saniye!!!\r\n";
61 /* USER CODE END 0 */
```

```
94 /* USER CODE BEGIN 2 */
95 NRF24_begin(nrf_CSN_PORT, nrf_CSN_PIN, nrf_CE_PIN, hspi1);
96 NRF24_stopListening();
97 NRF24_openWritingPipe(TxpipeAddrs);
98 NRF24_setAutoAck(false);
99 NRF24_setChannel(34);
100 /* USER CODE END 2 */
```

- MY_NRF24.h dosyasından ayrıntılara ulaşabiliriz.
- NRF24_begin başlatma fonksiyonudur.

```
204 void NRF24_begin(GPIO_TypeDef *nrf24PORT, uint16_t nrfCSN_Pin, uint16_t nrfCE_Pin, SPI_HandleTypeDef nrfSPI);
• Main.c dosyasında yazdığımız başlatma fonksiyondaki pinleri MY_NRF24.h kütüphanesinde tanımladık.
```

```
130 #define nrf_CSN_PORT      GPIOA
131 #define nrf_CSN_PIN       GPIO_PIN_2
132
133 #define nrf_CE_PORT      GPIOA
134 #define nrf_CE_PIN       GPIO_PIN_3
```

- Yazmak için Tx hattını açıyoruz. Dinleme işleminde bu işlemi yapamayız o yüzden öncesinde dinlemeyi

kapatmalıyız.

```
217 void NRF24_openWritingPipe(uint64_t address);
• Yazma işleminden önce bu fonksiyonu kullanarak CE pinini reset yapıyoruz.
208 void NRF24_stopListening(void);

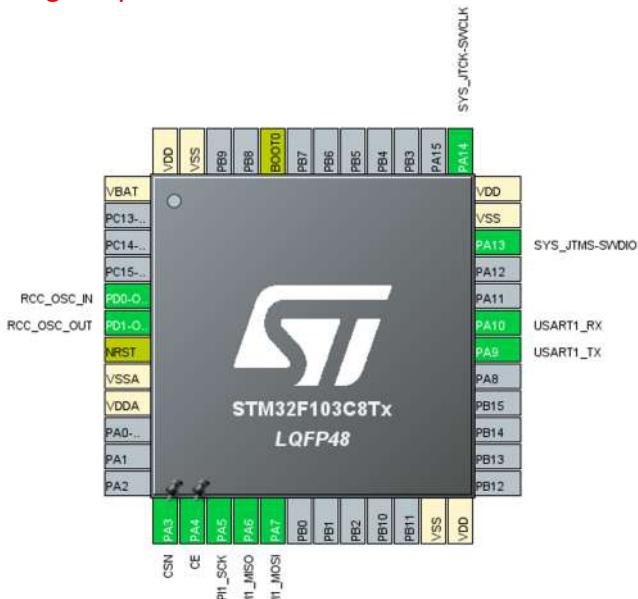
237 void NRF24_setAutoAck(bool enable);
Haberleşme için kanal frekansını yazıyoruz.
222 void NRF24_setChannel(uint8_t channel);
```

```
102 while (1)
103 {
104     /* USER CODE END WHILE */
105
106     /* USER CODE BEGIN 3 */
107     NRF24_write(myTxData, 32);
108     HAL_Delay(1000);
109 }
110 /* USER CODE END 3 */
111 }
```

- Veriyi yollamak için kullandığımız fonksiyon için yollayacağımız datayı ve datanın uzunluğunu yazıyoruz.
- ```
210 bool NRF24_write(const void* buf, uint8_t len);
```

## Alicı

### Konfigürasyon Kısmı



| Pin Name | Signal on Pin | GPIO output | GPIO mode          | GPIO Pull-up       | Maximum o... | User Label | Modified                            |
|----------|---------------|-------------|--------------------|--------------------|--------------|------------|-------------------------------------|
| PA5      | SPI1_SCK      | n/a         | Alternate Function | n/a                | High         |            | <input type="checkbox"/>            |
| PA6      | SPI1_MISO     | n/a         | Input mode         | No pull-up or down | n/a          |            | <input type="checkbox"/>            |
| PA7      | SPI1_MOSI     | n/a         | Alternate Function | n/a                | High         |            | <input type="checkbox"/>            |
| Pin Name | Signal on Pin | GPIO output | GPIO mode          | GPIO Pull-up       | Maximum o... | User Label | Modified                            |
| PA3      | n/a           | Low         | Output mode        | No pull-up or down | Low          | CSN        | <input checked="" type="checkbox"/> |
| PA4      | n/a           | n/a         | Analog input       | n/a                | n/a          | CE         | <input checked="" type="checkbox"/> |

- Connectivity Başlığı altında SPI1'i seçip Mode ekranından Full-Duplex Master'i seçiyoruz.

Mode  ▼

Hardware NSS Signal  ▼

| Basic Parameters                                                                                                                                                                                                   |               |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------|
| Frame Format                                                                                                                                                                                                       | Motorola      |
| Data Size                                                                                                                                                                                                          | 8 Bits        |
| First Bit                                                                                                                                                                                                          | MSB First     |
| Clock Parameters                                                                                                                                                                                                   |               |
| Prescaler (for Baud Rate)                                                                                                                                                                                          | 32            |
| Baud Rate                                                                                                                                                                                                          | 250.0 KBits/s |
| Clock Polarity (CPOL)                                                                                                                                                                                              | Low           |
| Clock Phase (CPHA)                                                                                                                                                                                                 | 1 Edge        |
| Advanced Parameters                                                                                                                                                                                                |               |
| CRC Calculation                                                                                                                                                                                                    | Disabled      |
| NSS Signal Type                                                                                                                                                                                                    | Software      |
| <ul style="list-style-type: none"> <li>• Connectivity Başlığı altında USART1'i seçip Mode ekranından Asynchronous'u seçiyoruz. Baud Rate değerininide 115200 yapıyoruz tabi farklı değerde seçebiliriz.</li> </ul> |               |
| Mode                                                                                                                                                                                                               | Asynchronous  |
| Hardware Flow Control (RS232)                                                                                                                                                                                      | Disable       |

| Basic Parameters    |                           |
|---------------------|---------------------------|
| Baud Rate           | 115200 Bits/s             |
| Word Length         | 8 Bits (including Parity) |
| Parity              | None                      |
| Stop Bits           | 1                         |
| Advanced Parameters |                           |
| Data Direction      | Receive and Transmit      |
| Over Sampling       | 16 Samples                |

### Kod Kismi

- MY\_NRF24.h dosyasında düzeltme yapıyoruz.

```

125 #include "stm32f4xx_hal.h"
• MY_NRF24.c dosyasına stdio.h kütüphanesini ekliyoruz.
21 #include "MY_NRF24.h"
22 #include "stdio.h"
• Kütüphanemizi dahil ettik.
23 /* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #include "MY_NRF24.h"
26 /* USER CODE END Includes */

58 /* Private user code -----
59 /* USER CODE BEGIN 0 */
60 uint64_t RxpipeAddrs = 122;
61 char RxData[50];
62 /* USER CODE END 0 */

94 /* USER CODE BEGIN 2 */
95 NRF24_begin(nrf_CSN_PORT, nrf_CSN_PIN, nrf_CE_PIN, hspi1);
96 nrf24_DebugUART_Init(huart1);
97 NRF24_setAutoAck(false);
98 NRF24_setChannel(34);
99 NRF24_openReadingPipe(1, RxpipeAddrs);
100 NRF24_startListening();
101 /* USER CODE END 2 */

```

- Main.c dosyasında yazdığımız başlatma fonksiyondaki pinleri MY\_NRF24.h kütüphanesinde tanımladık.

```

130 #define nrf_CSN_PORT GPIOA
131 #define nrf_CSN_PIN GPIO_PIN_3
132
133 #define nrf_CE_PORT GPIOA
134 #define nrf_CE_PIN GPIO_PIN_4
• Uart'ı tanımlıyoruz.
288 void nrf24_DebugUART_Init(UART_HandleTypeDef nrf24Uart);
• Okuma işlemi için açık hatları dinlemeyi başlatıyoruz. Öncesinde okuma hattını açmalıyız.
206 void NRF24_startListening(void);
• Okuma hattını açıyoruz.
219 void NRF24_openReadingPipe(uint8_t number, uint64_t address);

```

```
105 while (1)
106 {
107 /* USER CODE END WHILE */
108
109 /* USER CODE BEGIN 3 */
110 if(NRF24_available())
111 {
112 NRF24_read(RxData, 32);
113 HAL_UART_Transmit(&huart1, (uint8_t *)RxData, 32, 10);
114 }
115 } /* USER CODE END 3 */
```

- NRF24\_available ile okunacak veri olup olmadığını kontrol ediyoruz. Eğer veri gelirse alt satırda geçer.

```
213 bool NRF24_available(void);
```

Okumak için kullandığımız fonksiyon için okunacak datayı ve datanın uzunluğunu yazıyoruz.

```
310@bool NRF24_read(void* buf, uint8_t len)
```

# 14\_06 SD KART

9 Ocak 2022 Pazar 19:22

## [14\\_06 SD KART](#)

- <https://controllerstech.com/sd-card-using-spi-in-stm32/> adresinden örnek uygulamayı inceleyebiliriz.
- <https://youtu.be/teBa5-Ss1ww> linkinden yararlanabiliriz.

➤ [Veri Kaydetme](#)

**Konfigürasyon Küsmi**

**Kod Küsmi**

## 14\_07 GPS

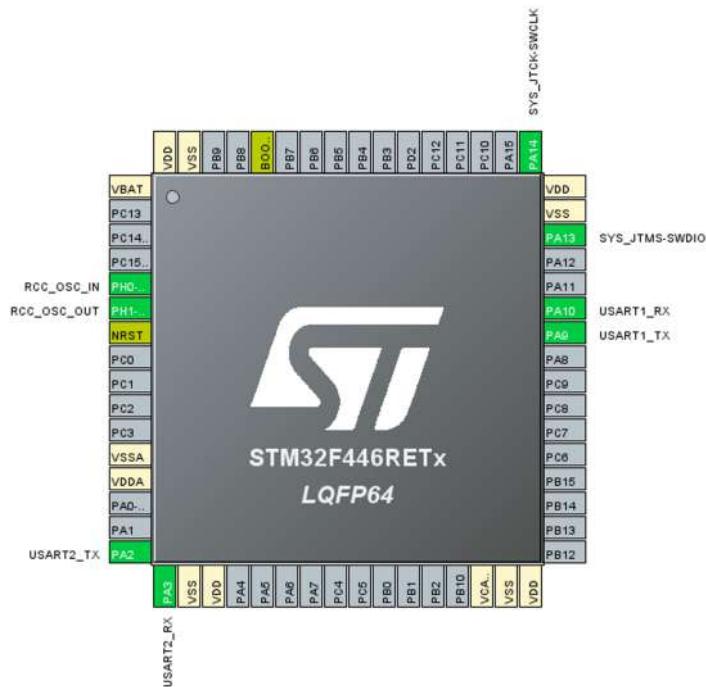
9 Ocak 2022 Pazar 19:22

### 14\_07 GPS

- <https://youtu.be/xJ18-6hNAkI> adresinden örnek uygulamayı inceleyebiliriz.
- NMEA protokolü hakkında bilgi edinmek için <https://ozcanfatih.wordpress.com/2016/07/24/nmea-0183-protokolu-ve-gps-verisi-ayiklama/> adresinden yararlanabiliriz.

#### ➤ Enlem ve Boylam

#### Konfigürasyon Kısımları



| Pin No. | Signal on Pin | GPIO out | GPIO mode | GPIO Pull  | Maximum   | User                     | Modified |
|---------|---------------|----------|-----------|------------|-----------|--------------------------|----------|
| PA2     | USART2_TX     | n/a      | Alternate | No pull-up | Very High | <input type="checkbox"/> |          |
| PA3     | USART2_RX     | n/a      | Alternate | No pull-up | Very High | <input type="checkbox"/> |          |

- USART1 için Asynchronous seçimi yapılır. Ardından parametre ayarlamaları yapılır.
- USART 1, GPS haberleşme için kullanıyoruz.

#### Basic Parameters

|             |                           |
|-------------|---------------------------|
| Baud Rate   | 9600 Bits/s               |
| Word Length | 8 Bits (including Parity) |
| Parity      | None                      |
| Stop Bits   | 1                         |

#### Advanced Parameters

|                |                      |
|----------------|----------------------|
| Data Direction | Receive and Transmit |
| Over Sampling  | 16 Samples           |

- USART1 için NVIC Settings kısmından Interrupt Enabled yapılır.

| NVIC Interrupt Table    | Enabled                             | Preemption Priority | Sub Priority |
|-------------------------|-------------------------------------|---------------------|--------------|
| USART2 global interrupt | <input checked="" type="checkbox"/> | 0                   | 0            |

- GPS için sadece receiver bacağını kullanacağız yani transmit bacağını kullanmayacağız.

#### Kod Kısımları

# 14\_08 LORA

9 Ocak 2022 Pazar 19:22

## 14\_08 LORA

- <https://github.com/Bob0505/E32-TTL-100> linkinden örnek uygulamaları inceleyebiliriz.

### ➤ Modül Çalıştırma

#### Verici

Konfigürasyon Kısmı

Kod Kısmı

#### Alıcı

Konfigürasyon Kısmı

Kod Kısmı

# 14\_09 BME280

24 Nisan 2022 Pazar 17:20

## 14\_09 BME280

- Kart hakkında bilgi edinmek için <https://learn.adafruit.com/adafruit-bme280-humidity-barometric-pressure-temperature-sensor-breakout> linki okunabilir.
- <https://controllerstech.com/bme280-with-stm32/> adresinden örnek uygulamayı inceleyebiliriz.
- Sensörün haberleşmelere göre bağlantı şékilleri aşağıdaki gibidir.

| Pin | Name              | I/O Type | Description                | Connect to        |         |                         |
|-----|-------------------|----------|----------------------------|-------------------|---------|-------------------------|
|     |                   |          |                            | SPI 4W            | SPI 3W  | I <sup>2</sup> C        |
| 1   | GND               | Supply   | Ground                     |                   | GND     |                         |
| 2   | CSB               | In       | Chip select                | CSB               | CSB     | V <sub>DDIO</sub>       |
| 3   | SDI               | In/Out   | Serial data input          | SDI               | SDI/SDO | SDA                     |
| 4   | SCK               | In       | Serial clock input         | SCK               | SCK     | SCL                     |
| 5   | SDO               | In/Out   | Serial data output         | SDO               | DNC     | GND for default address |
| 6   | V <sub>DDIO</sub> | Supply   | Digital / Interface supply | V <sub>DDIO</sub> |         |                         |
| 7   | GND               | Supply   | Ground                     | GND               |         |                         |
| 8   | V <sub>DD</sub>   | Supply   | Analog supply              | V <sub>DD</sub>   |         |                         |

| Register Name    | Address     | bit7        | bit6            | bit5             | bit4         | bit3        | bit2         | bit1 | bit0 | Reset state |
|------------------|-------------|-------------|-----------------|------------------|--------------|-------------|--------------|------|------|-------------|
| hum_lsb          | 0xFE        |             |                 |                  | hum_lsb<7:0> |             |              |      |      | 0x00        |
| hum_msb          | 0xFD        |             |                 |                  | hum_msb<7:0> |             |              |      |      | 0x80        |
| temp_xlsb        | 0xFC        |             | temp_xlsb<7:4>  |                  | 0            | 0           | 0            | 0    |      | 0x00        |
| temp_lsb         | 0xFB        |             |                 | temp_lsb<7:0>    |              |             |              |      |      | 0x00        |
| temp_msb         | 0xFA        |             |                 | temp_msb<7:0>    |              |             |              |      |      | 0x80        |
| press_xlsb       | 0xF9        |             | press_xlsb<7:4> |                  | 0            | 0           | 0            | 0    |      | 0x00        |
| press_lsb        | 0xF8        |             |                 | press_lsb<7:0>   |              |             |              |      |      | 0x00        |
| press_msb        | 0xF7        |             |                 | press_msb<7:0>   |              |             |              |      |      | 0x80        |
| config           | 0xF5        | t_sb[2:0]   |                 | filter[2:0]      |              |             | spi3w_en[0]  |      |      | 0x00        |
| ctrl_meas        | 0xF4        | osrs_t[2:0] |                 | osrs_p[2:0]      |              | mode[1:0]   |              |      |      | 0x00        |
| status           | 0xF3        |             |                 | measuring[0]     |              |             | im_update[0] |      |      | 0x00        |
| ctrl_hum         | 0xF2        |             |                 |                  |              | osrs_h[2:0] |              |      |      | 0x00        |
| calib26..calib41 | 0xE1...0xF0 |             |                 | calibration data |              |             |              |      |      | individual  |
| reset            | 0xE0        |             |                 | reset[7:0]       |              |             |              |      |      | 0x00        |
| id               | 0xD0        |             |                 | chip_id[7:0]     |              |             |              |      |      | 0x60        |
| calib00..calib25 | 0x88...0xA1 |             |                 | calibration data |              |             |              |      |      | individual  |

|            |                    |                  |                   |                |                  |           |            |
|------------|--------------------|------------------|-------------------|----------------|------------------|-----------|------------|
| Registers: | Reserved registers | Calibration data | Control registers | Data registers | Status registers | Chip ID   | Reset      |
| Type:      | do not change      | read only        | read / write      | read only      | read only        | read only | write only |

| Register 0xF2<br>"ctrl_hum" | Name        | Description                             |
|-----------------------------|-------------|-----------------------------------------|
| Bit 2, 1, 0                 | osrs_h[2:0] | Controls oversampling of humidity data. |

- ctrl\_hum register adındaki 0xF2 adresindeki ilk 3 bitteki osrs\_h[2:0] için nemin oversampling değeri kontrol edilir.
- Nem ölçümü için birçok oversampling seçeneği mevcuttur.

| osrs_h[2:0] | Humidity oversampling          |
|-------------|--------------------------------|
| 000         | Skipped (output set to 0x8000) |
| 001         | oversampling ×1                |
| 010         | oversampling ×2                |
| 011         | oversampling ×4                |
| 100         | oversampling ×8                |
| 101, others | oversampling ×16               |

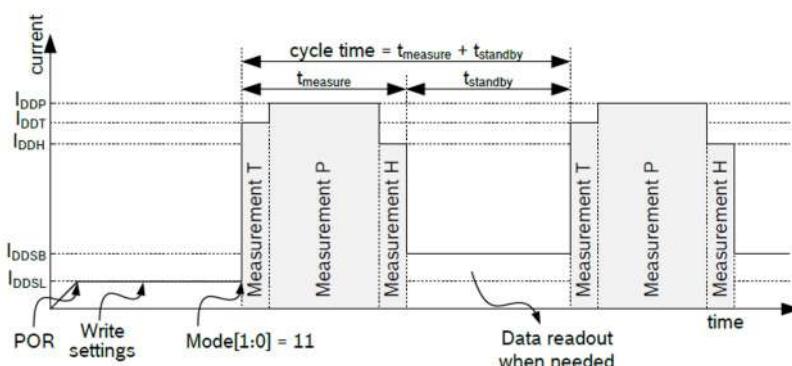
| Register 0xF4<br>"ctrl_meas" | Name        | Description                                |
|------------------------------|-------------|--------------------------------------------|
| Bit 7, 6, 5                  | osrs_t[2:0] | Controls oversampling of temperature data. |
| Bit 4, 3, 2                  | osrs_p[2:0] | Controls oversampling of pressure data.    |
| Bit 1, 0                     | mode[1:0]   | Controls the sensor mode of the device.    |

- ctrl\_meas register adındaki 0xF4 adresindeki ilk 2 bitteki mode[1:0] için cihazın sensör durumu , 2., 3. ve 4.bitte osrs\_p[2:0] için basıncın oversampling değeri, 5., 6. ve 7.bitte osrs\_t[2:0] için sıcaklığın oversampling değeri kontrol edilir.
- Basınç ve sıcaklık ölçümü için birçok oversampling seçeneği mevcuttur.
- Basınç ve sıcaklık ölçümünün çözünürlüğü IIRfiltresi ve oversampling ayarına bağlıdır. IIRfiltresi etkinleştirildiğinde 20bittir. IIRfiltresi devre dışı bırakıldığında  $16 + (\text{osrs}_p - 1)$  bittir.

| osrs_p[2:0] | Pressure oversampling           |
|-------------|---------------------------------|
| 000         | Skipped (output set to 0x80000) |
| 001         | oversampling ×1                 |
| 010         | oversampling ×2                 |
| 011         | oversampling ×4                 |
| 100         | oversampling ×8                 |
| 101, others | oversampling ×16                |

| osrs_t[2:0] | Temperature oversampling        |
|-------------|---------------------------------|
| 000         | Skipped (output set to 0x80000) |
| 001         | oversampling ×1                 |
| 010         | oversampling ×2                 |
| 011         | oversampling ×4                 |
| 100         | oversampling ×8                 |
| 101, others | oversampling ×16                |

- BME280'nin 3 güç modu bulunur. Bunlar uykı, normal ve zorunlu modlardır.
- Uykı modunda hiçbir ölçüm yapılmaz ve güç tüketimi minimumdur. Tüm kayıtlara erişilebilir ve bunları okuyabilir veya değiştirebiliriz.
- Zorunlu modda, sensör tek bir ölçüm yapar ve tekrar uykı moduna geçer. Ölçüm sonuçları veri kayıtlarından elde edilebilir. Bir sonraki ölçüm için, zorunlu modun yeniden seçilmesi gereklidir.
- Normal modda, uykı moduna geçmez. Ölçüm ve bekleme arasında devam eder. Bu mod genellikle, verilerin sürekli olarak gereklili olduğu durumlarda kullanılır.
- Normal modda toplam çevrim süresi, aktif sürenin ve bekleme süresinin toplamına bağlıdır.
- IIRfiltresini kullanırken normal modun kullanılması önerilir.



- Normal mode durumunda ölçüm süresi basınç, nem ve basınç için seçilen oversampling değerlerine bağlıdır. Aşağıdaki formül ile hesaplanarak ms cinsinden elde edilir.

$$t_{measure,typ} = 1 + [2 \cdot T_{oversampling}]_{osrs_t \neq 0} + [2 \cdot P_{oversampling} + 0.5]_{osrs_p \neq 0} + [2 \cdot H_{oversampling} + 0.5]_{osrs_h \neq 0}$$

$$t_{measure,max} = 1.25 + [2.3 \cdot T_{oversampling}]_{osrs_t \neq 0} + [2.3 \cdot P_{oversampling} + 0.575]_{osrs_p \neq 0} + [2.3 \cdot H_{oversampling} + 0.575]_{osrs_h \neq 0}$$

- Normal moddaki ölçüm hızı, ölçüm süresine ve bekleme süresine bağlıdır. Aşağıdaki formül ile hesaplanarak Hz cinsinden elde edilir.

$$ODR_{normal\_mode} = \frac{1000}{t_{measure} + t_{standby}}$$

| mode[1:0] | Mode        |
|-----------|-------------|
| 00        | Sleep mode  |
| 01 and 10 | Forced mode |
| 11        | Normal mode |

| Register 0xF5<br>"config" | Name        | Description                                              |
|---------------------------|-------------|----------------------------------------------------------|
| Bit 7, 6, 5               | t_sb[2:0]   | Controls inactive duration $t_{standby}$ in normal mode. |
| Bit 4, 3, 2               | filter[2:0] | Controls the time constant of the IIR filter.            |
| Bit 0                     | spi3w_en[0] | Enables 3-wire SPI interface when set to '1'.            |

- config register adındaki 0xF5 adresindeki ilk bitte spi3w\_en[0] kısmı, 2., 3. ve 4.bitte filter[2:0] için IIR filtre katsayısı, 5., 6. ve 7.bitte t\_sb[2:0] için normal mode durumundaki bekleme süresi kontrol edilir.
- Sıcaklık ve basınç için kısa süreli dalgalanmaları ortadan kaldırın isteğe bağlı IIR filtresinden geçirilir. Nem için böyle bir filtre gereklidir.

| t_sb[2:0] | t_standby [ms] |
|-----------|----------------|
| 000       | 0.5            |
| 001       | 62.5           |
| 010       | 125            |
| 011       | 250            |
| 100       | 500            |
| 101       | 1000           |
| 110       | 10             |
| 111       | 20             |

| filter[2:0] | Filter coefficient |
|-------------|--------------------|
| 000         | Filter off         |
| 001         | 2                  |
| 010         | 4                  |
| 011         | 8                  |
| 100, others | 16                 |

- press register adındaki 0xF7 ile 0xF9 arasındaki adreslerden basınç değeri okunur.

| Register 0xF7...0xF9<br>“press” | Name            | Description                                                                                                            |
|---------------------------------|-----------------|------------------------------------------------------------------------------------------------------------------------|
| 0xF7                            | press_msb[7:0]  | Contains the MSB part up[19:12] of the raw pressure measurement output data.                                           |
| 0xF8                            | press_lsb[7:0]  | Contains the LSB part up[11:4] of the raw pressure measurement output data.                                            |
| 0xF9 (bit 7, 6, 5, 4)           | press_xlsb[3:0] | Contains the XLSB part up[3:0] of the raw pressure measurement output data. Contents depend on temperature resolution. |

- temp register adındaki 0xFA ile 0xFC arasındaki adreslerden sıcaklık değeri okunur.

| Register 0xFA...0xFC<br>“temp” | Name           | Description                                                                                                            |
|--------------------------------|----------------|------------------------------------------------------------------------------------------------------------------------|
| 0xFA                           | temp_msb[7:0]  | Contains the MSB part ut[19:12] of the raw temperature measurement output data.                                        |
| 0xFB                           | temp_lsb[7:0]  | Contains the LSB part ut[11:4] of the raw temperature measurement output data.                                         |
| 0xFC (bit 7, 6, 5, 4)          | temp_xlsb[3:0] | Contains the XLSB part ut[3:0] of the raw temperature measurement output data. Contents depend on pressure resolution. |

- hum register adındaki 0xFD ile 0xFE arasındaki adreslerden nem değeri okunur.

| Register 0xFD...0xFE<br>“hum” | Name          | Description                                                                 |
|-------------------------------|---------------|-----------------------------------------------------------------------------|
| 0xFD                          | hum_msb[7:0]  | Contains the MSB part uh[15:8] of the raw humidity measurement output data. |
| 0xFE                          | temp_lsb[7:0] | Contains the LSB part uh[7:0] of the raw humidity measurement output data.  |

- Veriler, basınç ve sıcaklık için unsigned 20 bit formatında iken nem için unsigned 16 bit formatında okunur. Daha sonrasında okunan verilere kalibrasyon yapılmalıdır.

## ➤ Sıcaklık, Nem ve Basınç Okuma

### Konfigürasyon Kısımlı

| Pin N... | Signal on ... | GPIO out... | GPIO mo...    | GPIO Pull... | Maximum ... | User Label               | Modified |
|----------|---------------|-------------|---------------|--------------|-------------|--------------------------|----------|
| PB8      | I2C1_SCL      | n/a         | Alternate ... | Pull-up      | Very High   | <input type="checkbox"/> |          |
| PB9      | I2C1_SDA      | n/a         | Alternate ... | Pull-up      | Very High   | <input type="checkbox"/> |          |

- Connectivity başlığı altından I2C seçip Mode ekranından I2C seçiyoruz.

I2C I2C

- I2C için Speed Mode olarak Fast Mode seçimi yapıyoruz.

## Master Features

|                      |                           |
|----------------------|---------------------------|
| I2C Speed Mode       | Fast Mode                 |
| I2C Clock Speed (Hz) | 400000                    |
| Fast Mode Duty Cycle | Duty cycle Tlow/Thigh = 2 |

## Slave Features

|                              |          |
|------------------------------|----------|
| Clock No Stretch Mode        | Disabled |
| Primary Address Length       | 7-bit    |
| Dual Address Acknowledg.     | Disabled |
| Primary slave address        | 0        |
| General Call address detect. | Disabled |

## Kod Kısmı

- Dosyamızın Inc klasörü için BME280\_STM32.h adında Header File, Src klasörüne BME280\_STM32.c adında Source File ekliyoruz.
- Kütüphanelerde I2C kullanabilmek için main içerisinde tanımlanmış kütüphanemize ekliyoruz. Bunu tanımlayabilmek için öncesistm32f4xx\_hal.h kutuphanesini dahil edildiğinden emin oluyoruz.
- BME280\_STM32.h dosyasına aşağıdaki satırı kontrol ediyoruz.

25 `#include "stm32f4xx_hal.h"`

- BME280\_STM32.c dosyasında I2C tanımının eklendiğini görüyoruz.

23 `extern I2C_HandleTypeDef hi2c1;`

24 `#define BME280_I2C &hi2c1`

- 26. ile 27.satırda birini seçiyoruz. Biz 26.satırı kullanıyoruz.

26 `#define SUPPORT_64BIT 1`

27 `//#define SUPPORT_32BIT 1`

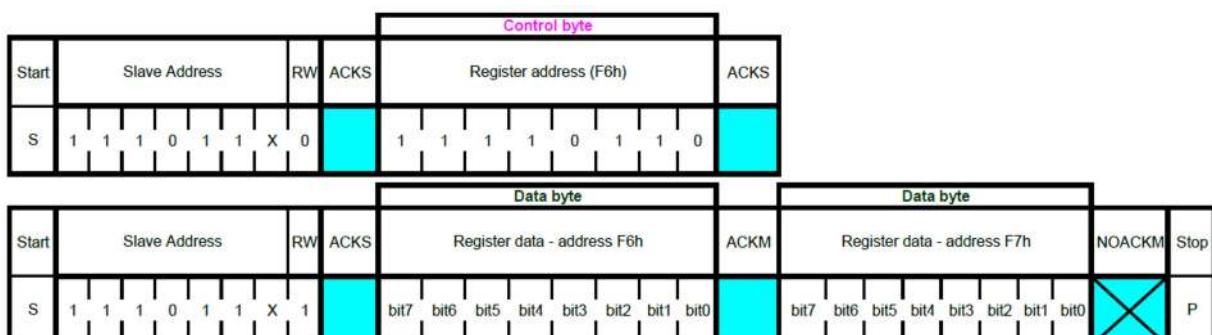
- Ardından slave adresi tanımlıyoruz.
- 7 bitlik cihaz adresi 111011X şeklinde dir. Buradaki X, SDO pinin durumuna göre belirlenir. SDO'nun GND'ye bağlanması durumunda X=0 olur böylece 1110110 yani 0x76 adreslidir. SDO'nun VDDIO'ya bağlanması durumunda X=1 olur böylece 1110111 yani 0x77 adreslidir. Bunları 8 bit yapmak için 1 bit sola kaydırılırsa 0x76 adresi 0xEC, 0x77 adresi 0xEE olur.
- SDO pinini GND'ye bağlıyoruz böylece slave adresimiz 0xEC olur.

29 `#define BME280_ADDRESS 0xEC // SDO is GND, 7 bit address is 0x76, 8 bit address 0x76<<1 = 0xEC`

- I2C yazma modunda iken RW=0 olur.



- I2C okuma moduna geçebilmek için önce yazma modu gönderilir. I2C okuma modunda iken RW=1 olur.



- Main.c içerisinde tanımladığımız değişkenleri BME280\_STM32.c dosyasına ekleriz.

31 `extern float Temperature, Pressure, Humidity;`

- BME280\_STM32.h dosyasında tanımlanan aşağıdadır.

```

27/* Configuration for the BME280
28
29 * @osrs is the oversampling to improve the accuracy
30 * if osrs is set to OSRS_OFF, the respective measurement will be skipped
31 * It can be set to OSRS_1, OSRS_2, OSRS_4, etc. Check the header file
32 *
33 * @mode can be used to set the mode for the device
34 * MODE_SLEEP will put the device in sleep
35 * MODE_FORCED device goes back to sleep after one measurement. You need to use the BME280_WakeUP() function before every measurement
36 * MODE_NORMAL device performs measurement in the normal mode. Check datasheet page no 16
37 *
38 * @t_sb is the standby time. The time sensor waits before performing another measurement
39 * It is used along with the normal mode. Check datasheet page no 16 and page no 30
40 *
41 * @filter is the IIR filter coefficients
42 * IIR is used to avoid the short term fluctuations
43 * Check datasheet page no 18 and page no 30
44 */
45
46 int BME280_Config (uint8_t osrs_t, uint8_t osrs_p, uint8_t osrs_h, uint8_t mode, uint8_t t_sb, uint8_t filter);
47
48 // Read the Trimming parameters saved in the NVM ROM of the device
49 void TrimRead(void);
50
51/* To be used when doing the force measurement
52 * the Device need to be put in forced mode every time the measurement is needed
53 */
54 void BME280_WakeUP(void);
55
56/* measure the temp, pressure and humidity
57 * the values will be stored in the parameters passed to the function
58 */
59 void BME280_Measure (void);

63 // Oversampling definitions
64 #define OSRS_OFF 0x00
65 #define OSRS_1 0x01
66 #define OSRS_2 0x02
67 #define OSRS_4 0x03
68 #define OSRS_8 0x04
69 #define OSRS_16 0x05
70
71 // MODE Definitions
72 #define MODE_SLEEP 0x00
73 #define MODE_FORCED 0x01
74 #define MODE_NORMAL 0x03
75
76 // Standby Time
77 #define T_SB_0p5 0x00
78 #define T_SB_62p5 0x01
79 #define T_SB_125 0x02
80 #define T_SB_250 0x03
81 #define T_SB_500 0x04
82 #define T_SB_1000 0x05
83 #define T_SB_10 0x06
84 #define T_SB_20 0x07
85
86 // IIR Filter Coefficients
87 #define IIR_OFF 0x00
88 #define IIR_2 0x01
89 #define IIR_4 0x02
90 #define IIR_8 0x03
91 #define IIR_16 0x04
92
93 // REGISTERS DEFINITIONS
94 #define ID_REG 0xD0
95 #define RESET_REG 0xE0
96 #define CTRL_HUM_REG 0xF2
97 #define STATUS_REG 0xF3
98 #define CTRL_MEAS_REG 0xF4
99 #define CONFIG_REG 0xF5
100 #define PRESS_MSB_REG 0xF7

```

- BME280\_WakeUP() fonksiyonu cihazın mod durumunu okur ardından modu Zorunlu mod olarak değiştirir.

```

184/* To be used when doing the force measurement
185 * the Device need to be put in forced mode every time the measurement is needed
186 */
187void BME280_WakeUP(void)
188{
189 uint8_t datatowrite = 0;
190
191 // first read the register
192 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, CTRL_MEAS_REG, 1, &datatowrite, 1, 1000);
193
194 // modify the data with the forced mode
195 datatowrite = datatowrite | MODE_FORCED;
196
197 // write the new data to the register
198 HAL_I2C_Mem_Write(BME280_I2C, BME280_ADDRESS, CTRL_MEAS_REG, 1, &datatowrite, 1, 1000);
199
200 HAL_Delay (100);
201 }

```

- Datasheet'te belirtilen Trimming parametreleri okunur.

| Register Address | Register content      | Data type      |
|------------------|-----------------------|----------------|
| 0x88 / 0x89      | dig_T1 [7:0] / [15:8] | unsigned short |
| 0x8A / 0x8B      | dig_T2 [7:0] / [15:8] | signed short   |
| 0x8C / 0x8D      | dig_T3 [7:0] / [15:8] | signed short   |
| 0x8E / 0x8F      | dig_P1 [7:0] / [15:8] | unsigned short |
| 0x90 / 0x91      | dig_P2 [7:0] / [15:8] | signed short   |
| 0x92 / 0x93      | dig_P3 [7:0] / [15:8] | signed short   |
| 0x94 / 0x95      | dig_P4 [7:0] / [15:8] | signed short   |
| 0x96 / 0x97      | dig_P5 [7:0] / [15:8] | signed short   |
| 0x98 / 0x99      | dig_P6 [7:0] / [15:8] | signed short   |
| 0x9A / 0x9B      | dig_P7 [7:0] / [15:8] | signed short   |
| 0x9C / 0x9D      | dig_P8 [7:0] / [15:8] | signed short   |
| 0x9E / 0x9F      | dig_P9 [7:0] / [15:8] | signed short   |
| 0xA1             | dig_H1 [7:0]          | unsigned char  |
| 0xE1 / 0xE2      | dig_H2 [7:0] / [15:8] | signed short   |
| 0xE3             | dig_H3 [7:0]          | unsigned char  |
| 0xE4 / 0xE5[3:0] | dig_H4 [11:4] / [3:0] | signed short   |
| 0xE5[7:4] / 0xE6 | dig_H5 [3:0] / [11:4] | signed short   |
| 0xE7             | dig_H6                | signed char    |

- Verilerin değişkenlerini yazıyoruz.

```

38 uint16_t dig_T1, \
39 dig_P1, \
40 dig_H1, dig_H3;
41
42 int16_t dig_T2, dig_T3, \
43 dig_P2, dig_P3, dig_P4, dig_P5, dig_P6, dig_P7, dig_P8, dig_P9, \
44 dig_H2, dig_H4, dig_H5, dig_H6;

```

- HAL\_I2C\_Mem\_Read fonksiyonuyla slave cihazdan okuduğumuz verileri diziye yazıyoruz. Ardından dizedeki verileri atadığımız değişkenlere yazıyoruz.

```

48 // Read the Trimming parameters saved in the NVM ROM of the device
49 void TrimRead(void)
50 {
51 uint8_t trimdata[32];
52 // Read NVM from 0x88 to 0xA1
53 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, 0x88, 1, trimdata, 25, HAL_MAX_DELAY);
54
55 // Read NVM from 0xE1 to 0xE7
56 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, 0xE1, 1, (uint8_t *)trimdata+25, 7, HAL_MAX_DELAY);
57
58 // Arrange the data as per the datasheet (page no. 24)
59 dig_T1 = (trimdata[1]<<8) | trimdata[0];
60 dig_T2 = (trimdata[3]<<8) | trimdata[2];
61 dig_T3 = (trimdata[5]<<8) | trimdata[4];
62 dig_P1 = (trimdata[7]<<8) | trimdata[5];
63 dig_P2 = (trimdata[9]<<8) | trimdata[6];
64 dig_P3 = (trimdata[11]<<8) | trimdata[10];
65 dig_P4 = (trimdata[13]<<8) | trimdata[12];
66 dig_P5 = (trimdata[15]<<8) | trimdata[14];
67 dig_P6 = (trimdata[17]<<8) | trimdata[16];
68 dig_P7 = (trimdata[19]<<8) | trimdata[18];
69 dig_P8 = (trimdata[21]<<8) | trimdata[20];
70 dig_P9 = (trimdata[23]<<8) | trimdata[22];
71 dig_H1 = trimdata[24];
72 dig_H2 = (trimdata[26]<<8) | trimdata[25];
73 dig_H3 = (trimdata[27]);
74 dig_H4 = (trimdata[28]<<4) | (trimdata[29] & 0x0f);
75 dig_H5 = (trimdata[30]<<4) | (trimdata[29]>>4);
76 dig_H6 = (trimdata[31]);
77 }

```

- 0xF7 ile 0xFE arasındaki adreslerdeki basınç, sıcaklık ve nem değerlerini okuyoruz.
- Okumadan önce id kontrol edilir.

```

33 uint8_t chipID;
• id register adındaki 0xDO adresinde çip kimlik numarası olarak 0x60 içermesi lazım bu yüzden
 0XDO adresinden okuma yapıp chipID'ye yazıyoruz. Eğer 0x60 ile eşleşirse okuma yapabiliriz.
36 int32_t tRaw, pRaw, hRaw;

```

```

159 int BMEReadRaw(void)
160 {
161 uint8_t RawData[8];
162
163 // Check the chip ID before reading
164 HAL_I2C_Mem_Read(&hi2c1, BME280_ADDRESS, ID_REG, 1, &chipID, 1, 1000);
165
166 if (chipID == 0x60)
167 {
168 // Read the Registers 0xF7 to 0xFE
169 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, PRESS_MSB_REG, 1, RawData, 8, HAL_MAX_DELAY);
170
171 /* Calculate the Raw data for the parameters
172 * Here the Pressure and Temperature are in 20 bit format and humidity in 16 bit format
173 */
174 pRaw = (RawData[0]<<12)|(RawData[1]<<4)|(RawData[2]>>4);
175 tRaw = (RawData[3]<<12)|(RawData[4]<<4)|(RawData[5]>>4);
176 hRaw = (RawData[6]<<8)|(RawData[7]);
177
178 return 0;
179 }
180
181 else return -1;
182 }

```

```

205 /* Returns temperature in DegC, resolution is 0.01 DegC. Output value of "5123" equals 51.23 DegC.
206 t_fine carries fine temperature as global value
207 */
208 int32_t t_fine;
209 int32_t BME280_compensate_T_int32(int32_t adc_T)
210 {
211 int32_t var1, var2, T;
212 var1 = (((adc_T>>3) - ((int32_t)dig_T1<<1)) * ((int32_t)dig_T2)) >> 11;
213 var2 = (((adc_T>>4) - ((int32_t)dig_T1)) * ((adc_T>>4) - ((int32_t)dig_T1)))>> 12) *((int32_t)dig_T3)) >> 14;
214 t_fine = var1 + var2;
215 T = (t_fine * 5 + 128) >> 8;
216 return T;
217 }

```

```

220 /* Returns pressure in Pa as unsigned 32 bit integer in Q24.8 format (24 integer bits and 8 fractional bits).
221 Output value of "24674867" represents 24674867/256 = 96386.2 Pa = 963.862 hPa
222 */
223 uint32_t BME280_compensate_P_int64(int32_t adc_P)
224 {
225 int64_t var1, var2, p;
226 var1 = ((int64_t)t_fine) - 128000;
227 var2 = var1 * var1 * (int64_t)dig_P6;
228 var2 = var2 + ((var1*(int64_t)dig_P5)<<17);
229 var2 = var2 + (((int64_t)dig_P4)<<35);
230 var1 = ((var1 * var1 * (int64_t)dig_P3)>>8) + ((var1 * (int64_t)dig_P2)<<12);
231 var1 = (((((int64_t)1)<<47)+var1)*((int64_t)dig_P1)>>33;
232 if (var1 == 0)
233 {
234 return 0; // avoid exception caused by division by zero
235 }
236 p = 1048576-adc_P;
237 p = (((p<<31)-var2)*3125)/var1;
238 var1 = (((int64_t)dig_P9) * (p>>13) * (p>>13)) >> 25;
239 var2 = (((int64_t)dig_P8) * p) >> 19;
240 p = ((p + var1 + var2) >> 8) + (((int64_t)dig_P7)<<4);
241 return (uint32_t)p;
242 }

```

- Basınç için,

```

276 /* Returns humidity in %RH as unsigned 32 bit integer in Q22.10 format (22 integer and 10 fractional bits).
277 Output value of "47445" represents 47445/1024 = 46.333 %RH
278 */
279 uint32_t bme280_compensate_H_int32(int32_t adc_H)
280 {
281 int32_t v_x1_u32r;
282 v_x1_u32r = (t_fine - ((int32_t)76800));
283 v_x1_u32r = (((((adc_H << 14) - ((int32_t)dig_H4) << 20) - (((int32_t)dig_H5) *\
284 v_x1_u32r)) + ((int32_t)16384)) >> 15) * (((((v_x1_u32r *\
285 ((int32_t)dig_H6)) >> 10) * (((v_x1_u32r * ((int32_t)dig_H3)) >> 11) +\
286 ((int32_t)32768)) >> 10) + ((int32_t)2097152)) * ((int32_t)dig_H2) +\
287 8192) >> 14));
288 v_x1_u32r = (v_x1_u32r - (((((v_x1_u32r >> 15) * (v_x1_u32r >> 15)) >> 7) *\
289 ((int32_t)dig_H1)) >> 4));
290 v_x1_u32r = (v_x1_u32r < 0 ? 0 : v_x1_u32r);
291 v_x1_u32r = (v_x1_u32r > 419430400 ? 419430400 : v_x1_u32r);
292 return (uint32_t)(v_x1_u32r>>12);
293 }

```

- TrimRead() fonksiyonunu çağırıyoruz. Okunan değerler kalıcı bellekte değişmeden saklanır.
- Okunduktan sonra cihazı resetleriz. Bunun için reset register adındaki 0xE0 adresine 0xB6 yazmamız durumunda cihaz açılışta sıfırlanır. 0xB6 dışında bir şey yazmanın bir etkisi yoktur. 0xE0 adresinin okuma değeri her zaman 0x00'dır.
- Sıfırladıktan sonra Nem için oversampling değerini 0xF2 adresine yazıyoruz.

- Ardından bekleme süresi ile filtre katsayısını 0xF5 adresine yazıyoruz. Bekleme süresini yazarken 5 bit, filtre katsayısını 2 bit sola kaydırıyorum.
- Daha sonra sıcaklık ile basınç için oversampling değerini ve mode durumunu 0xF4 adresine yazıyoruz. Sıcaklığın oversampling değerini yazarken 5 bit, basınçın oversampling değerini 2 bit sola kaydırıyorum. Modu aynen yazıyorum.

```

78/* Configuration for the BME280
79
80 * @osrs is the oversampling to improve the accuracy
81 * if osrs is set to OSRS_OFF, the respective measurement will be skipped
82 * It can be set to OSRS_1, OSRS_2, OSRS_4, etc. Check the header file
83 *
84 * @mode can be used to set the mode for the device
85 * MODE_SLEEP will put the device in sleep
86 * MODE_FORCED device goes back to sleep after one measurement. You need to use the BME280_WakeUP() function before every measurement
87 * MODE_NORMAL device performs measurement in the normal mode. Check datasheet page no 16
88 *
89 * @t_sb is the standby time. The time sensor waits before performing another measurement
90 * It is used along with the normal mode. Check datasheet page no 16 and page no 30
91 *
92 * @filter is the IIR filter coefficients
93 * IIR is used to avoid the short term fluctuations
94 * Check datasheet page no 18 and page no 30
95 */
96
97int BME280_Config (uint8_t osrs_t, uint8_t osrs_p, uint8_t osrs_h, uint8_t mode, uint8_t t_sb, uint8_t filter)
98{
99 // Read the Trimming parameters
100 TrimRead();
101
102 uint8_t datatowrite = 0;
103 uint8_t datacheck = 0;
104
105 // Reset the device
106 datatowrite = 0xB6; // reset sequence
107 if (HAL_I2C_Mem_Write(BME280_I2C, BME280_ADDRESS, RESET_REG, 1, &datatowrite, 1, 1000) != HAL_OK)
108 {
109 return -1;
110 }
111 HAL_Delay (100);
112
113
114 // write the humidity oversampling to 0xF2
115 datatowrite = osrs_h;
116 if (HAL_I2C_Mem_Write(BME280_I2C, BME280_ADDRESS, CTRL_HUM_REG, 1, &datatowrite, 1, 1000) != HAL_OK)
117 {
118 return -1;
119 }
120 HAL_Delay (100);
121 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, CTRL_HUM_REG, 1, &datacheck, 1, 1000);
122 if (datacheck != datatowrite)
123 {
124 return -1;
125 }
126
127
128 // write the standby time and IIR filter coeff to 0xF5
129 datatowrite = (t_sb <<5) |(filter << 2);
130 if (HAL_I2C_Mem_Write(BME280_I2C, BME280_ADDRESS, CONFIG_REG, 1, &datatowrite, 1, 1000) != HAL_OK)
131 {
132 return -1;
133 }
134 HAL_Delay (100);
135 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, CONFIG_REG, 1, &datacheck, 1, 1000);
136 if (datacheck != datatowrite)
137 {
138 return -1;
139 }
140
141
142 // write the pressure and temp oversampling along with mode to 0xF4
143 datatowrite = (osrs_t <<5) |(osrs_p << 2) | mode;
144 if (HAL_I2C_Mem_Write(BME280_I2C, BME280_ADDRESS, CTRL_MEAS_REG, 1, &datatowrite, 1, 1000) != HAL_OK)
145 {
146 return -1;
147 }
148 HAL_Delay (100);
149 HAL_I2C_Mem_Read(BME280_I2C, BME280_ADDRESS, CTRL_MEAS_REG, 1, &datacheck, 1, 1000);
150 if (datacheck != datatowrite)
151 {
152 return -1;
153 }
154
155 return 0;
156 }
```

- Artık main.c dosyasına kodumuzu yazmaya başlayabiliriz.
- Kütüphane eklenir.

```

23/* Private includes -----
24 /* USER CODE BEGIN Includes */
```

```

23/* Private includes -----
24 /* USER CODE BEGIN Includes */
25 #include "BME280_STM32.h"
26 /* USER CODE END Includes */

```

- Değişkenler tanımlanır.

```

57/* Private user code -----
58 /* USER CODE BEGIN 0 */
59 float Temperature, Pressure, Humidity;
60 /* USER CODE END 0 */

```

- BME280\_Config() fonksiyonun parametreleri girilir.
- Sıcaklık için oversampling 2, basınç için oversampling 16, nem için oversampling 1, mode olarak normal mode, bekleme süresi 0.5ms, ve filtre katsayısını 16 giriyoruz.
- Bu değerleri girmek için datasheette örnek çalışma modları var. Biz Indoor navigation yani iç mekan navigasyon örneğini kullanıyoruz.

| Suggested settings for indoor navigation |                                           |
|------------------------------------------|-------------------------------------------|
| Sensor mode                              | normal mode, tstandby = 0.5 ms            |
| Oversampling settings                    | pressure ×16, temperature ×2, humidity ×1 |
| IIR filter settings                      | filter coefficient 16                     |
| Performance for suggested settings       |                                           |
| Current consumption                      | 633 µA                                    |
| RMS Noise                                | 0.2 Pa / 1.7 cm                           |
| Data output rate                         | 25Hz                                      |
| Filter bandwidth                         | 0.53 Hz                                   |
| Response time (75%)                      | 0.9 s                                     |

```

91 /* USER CODE BEGIN 2 */
92 BME280_Config(OSRS_2, OSRS_16, OSRS_1, MODE_NORMAL, T_SB_0p5, IIR_16);
93 /* USER CODE END 2 */

```

- Döngümüzde BME280\_Measure() fonksiyonunu çağırırız.

```

95 /* Infinite loop */
96 /* USER CODE BEGIN WHILE */
97 while (1)
98 {
99 /* USER CODE END WHILE */
100
101 /* USER CODE BEGIN 3 */
102 BME280_Measure();
103 HAL_Delay (500);
104 }
105 /* USER CODE END 3 */
106 }

```

- Fonksiyonun içeriği aşağıdadır.
- Nem birimi %RH, basınç birimi Pa, sıcaklık birimi DegC'dir.

```

297/* measure the temp, pressure and humidity
298 * the values will be stored in the parameters passed to the function
299 */
300void BME280_Measure (void)
301{
302 if (BMEReadRaw() == 0)
303 {
304 if (tRaw == 0x800000) Temperature = 0; // value in case temp measurement was disabled
305 else
306 {
307 Temperature = (BME280_compensate_T_int32 (tRaw))/100.0; // as per datasheet, the temp is ×100
308 }
309
310 if (pRaw == 0x800000) Pressure = 0; // value in case temp measurement was disabled
311 else
312 {
313#if SUPPORT_64BIT
314 Pressure = (BME280_compensate_P_int64 (pRaw))/256.0; // as per datasheet, the pressure is ×256
315
316 if (hRaw == 0x8000) Humidity = 0; // value in case temp measurement was disabled
317 else
318 {
319 Humidity = (bme280_compensate_H_int32 (hRaw))/1024.0; // as per datasheet, the temp is ×1024
320 }
321 }
322
323 // if the device is detached
324 else
325 {
326 Temperature = Pressure = Humidity = 0;
327 }
328 }
329}

```

- ırtifa verisini ekleme için main.c dosyasında Altitude değişkeni tanımlıyorum.

```
58 /* USER CODE BEGIN 0 */
59 float Temperature, Pressure, Humidity, Altitude;
60 /* USER CODE END 0 */
```

- Altitude değişkenini BME280\_STM32.c dosyasına ekliyorum.

```
32 extern float Temperature, Pressure, Humidity, Altitude;
```

- Ardından irtifa formülünde kullanacağım pow matematik değeri için math.h kütüphanesini ekliyorum.

```
22 #include "math.h"
```

- BME280\_Measure() fonksiyonunda basıncı bulduğum satırın altına irtifa formülünü yazıyorum.

```
316 Altitude = 44330*(1-(pow((float)Pressure/(float)101325), 0.19029495718)));
```

- BME280 sensöründen okuma yaptığımız basınç, sıcaklık ve nem verisi ile basınç verisini kullanarak elde ettiğimiz irtifa verisinin sonuçları aşağıdadır.

| Variable Name | Address/Expression | Read Value |
|---------------|--------------------|------------|
| Altitude      | 0x20000148         | 4.831126   |
| Humidity      | 0x20000150         | 54.82129   |
| Pressure      | 0x2000014c         | 101266.984 |
| Temperature   | 0x200000f0         | 24.97      |

# 14\_10 BMP280

2 Mayıs 2022 Pazartesi 05:05

## 14\_10 BMP280

- <https://youtu.be/WGkx65aDJcw> ile <https://youtu.be/TNWwas6Zh7I> linkindeki örnek uygulamayı inceleyebiliriz.

➤ Sıcaklık ve Basınç Okuma

Konfigürasyon Kısmı

Kod Kısmı