I2C Kullanımı

25 Aralık 2021 Cumartesi 01:02

I2C Kullanımı

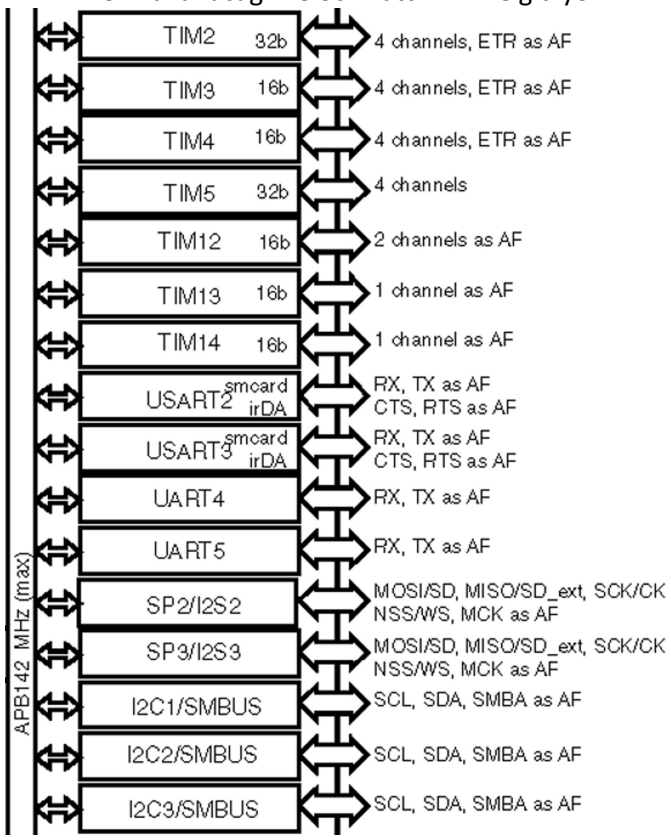
➢ REGİSTER

Konfigürasyon Kısmı

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
void RCC_Config(void)
{
    RCC->CR |= 0x00010000;              //HSEON
    while(!(RCC->CR & 0x00020000));     //HSERDY
    RCC->CR |= 0x00080000;              //CSSON
    RCC->CFGR = 0x00000000;
    RCC->PLLCFGR |= 0x00400000;         //PLLSRC
    RCC->PLLCFGR |= 0x00000004;         //PLLM 4
    RCC->PLLCFGR |= 0x00002A00;         //PLLN 168
    RCC->PLLCFGR |= 0x00000000;         //PLLP 2
    RCC->CR |= 0x01000000;              //PLLON
    while(!(RCC->CR & 0x02000000));     //PLLRDY
    RCC->CFGR |= 0x00000001;            //SW
    while(!(RCC->CR & 0x00000001));     //SWS
    RCC->CFGR |= 0x00000000;            //HPRE AHB 1
    RCC->CFGR |= 0x00001400;            //PPRE1 APB1 4
    RCC->CFGR |= 0x00008000;            //PPRE2 APB2 2
    RCC->CIR |= 0x00080000;             //HSERDYC
    RCC->CIR |= 0x00800000;             //CSSC
}
```

- I2C2 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	

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

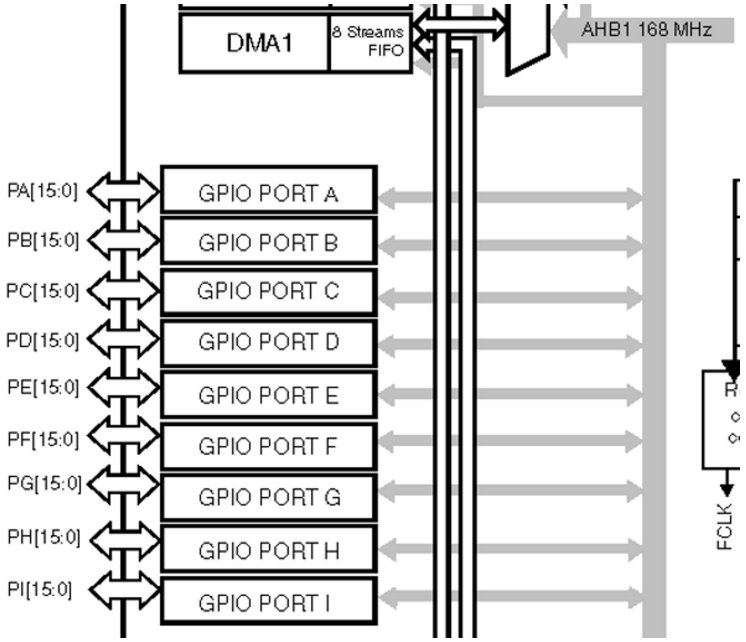
1: I2C2 clock enabled

```
RCC->APB1ENR |= 1 << 22;
```

- I2C2 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B portun 10. ve 11.pinine bağlıymış.

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/I2S2e xt	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_ SMBA	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 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	OTGHS ULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved		DMA2EN	DMA1EN	CCMDATARAMEN	Res.	BKPSRAMEN	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			CRCEN	Reserved			GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN
			rw				rw	rw	rw	rw	rw	rw	rw	rw	rw

- 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 enabled

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 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

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]	

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 |= (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 **AFRHy:** Alternate function selection for port x bit y (y = 8..15)

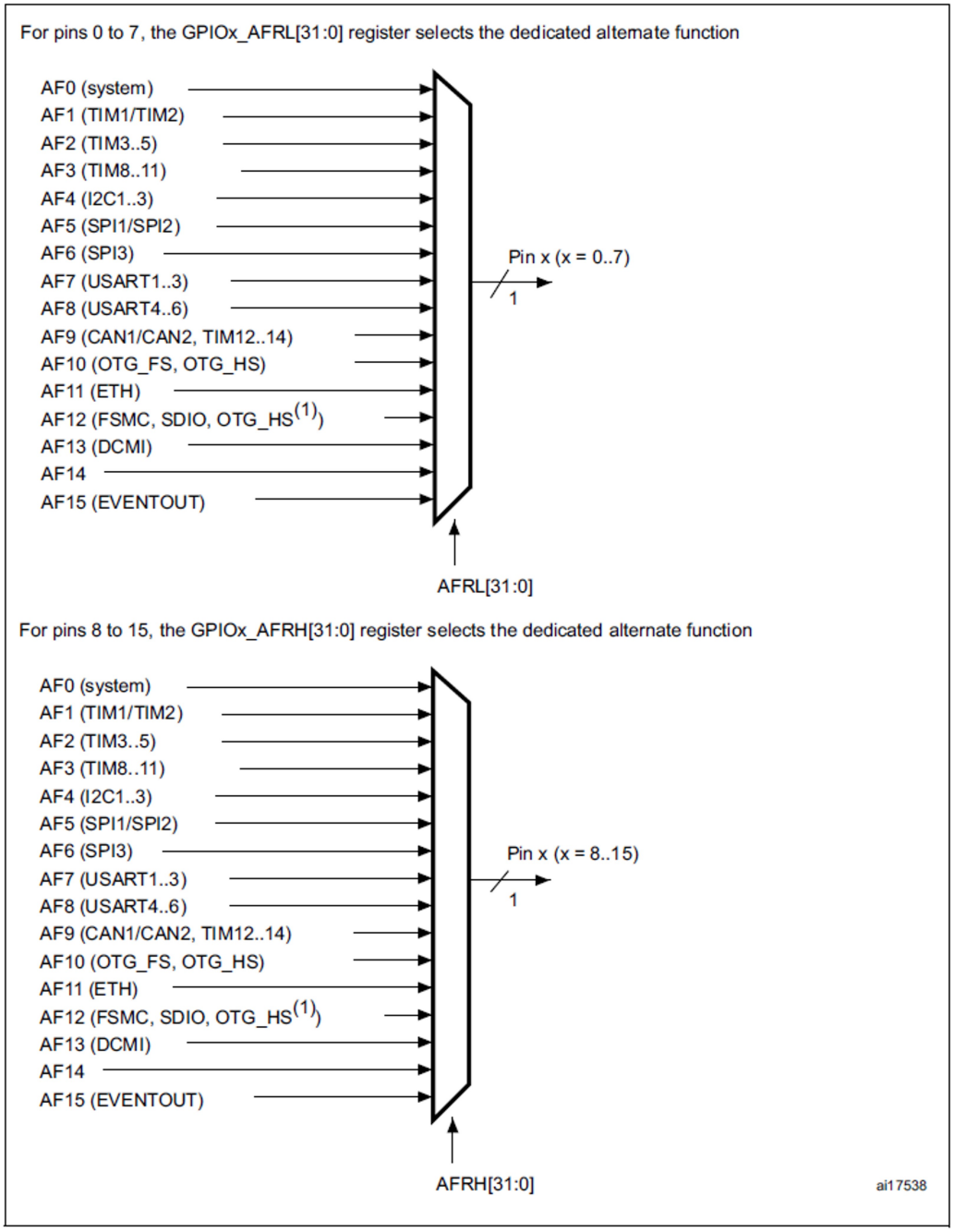
These bits are written by software to configure alternate function I/Os

AFRHy 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 kitapçığına bakıyoruz.

Selecting an alternate function



- I2C, AF7'de bulunuyor. Böylece pinlere AF4 için olan 0100 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunuda 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

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

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

$t_{high} = t_{r(SCL)} + t_{w(SCLH)}$. 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, fSCL ~ 1/(thigh + tlow). 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 demiş.

```
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 adress 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

- Yolladığım 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 "!" işareti koyarak tersliyoruz. 0.bit 1 olduğunda döngüden çıkacaktır.

```
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	ENGC	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 adress
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;
54     while(!(I2C2->SR2 & (1 << 2)));         //Data byte transfer succeeded
55     I2C2->CR1 |= (1 << 9);                  //Stop generation
56 }
```

- Her butona bastığımızda cihaza sırayla adres yolluyor.

```c
int main(void)
{
    RCC_Config();
    GPIO_Config();
    I2C_Config();

  while (1)
  {
      if(GPIOA->IDR & 0x00000001)
          {
              i++;
              delay(6300000);
          }

          switch(i)
          {
          case 0:
              I2C_Write(m_address, 0x00);
              break;
          case 1:
              I2C_Write(m_address, 0x01);
              break;
          case 2:
              I2C_Write(m_address, 0x02);
              break;
          case 3:
              I2C_Write(m_address, 0x04);
              break;
          case 4:
              I2C_Write(m_address, 0x08);
              break;
          case 5:
              I2C_Write(m_address, 0x10);
              break;
          case 6:
              I2C_Write(m_address, 0x20);
              break;
          case 7:
              I2C_Write(m_address, 0x40);
              break;
          case 8:
              I2C_Write(m_address, 0x80);
              break;
          default:
              i=0;
              break;
          }

  }
}
```