

UART ile Mesaj Gönderme

16 Haziran 2023 Cuma 10:44

UART ile Mesaj Gönderme

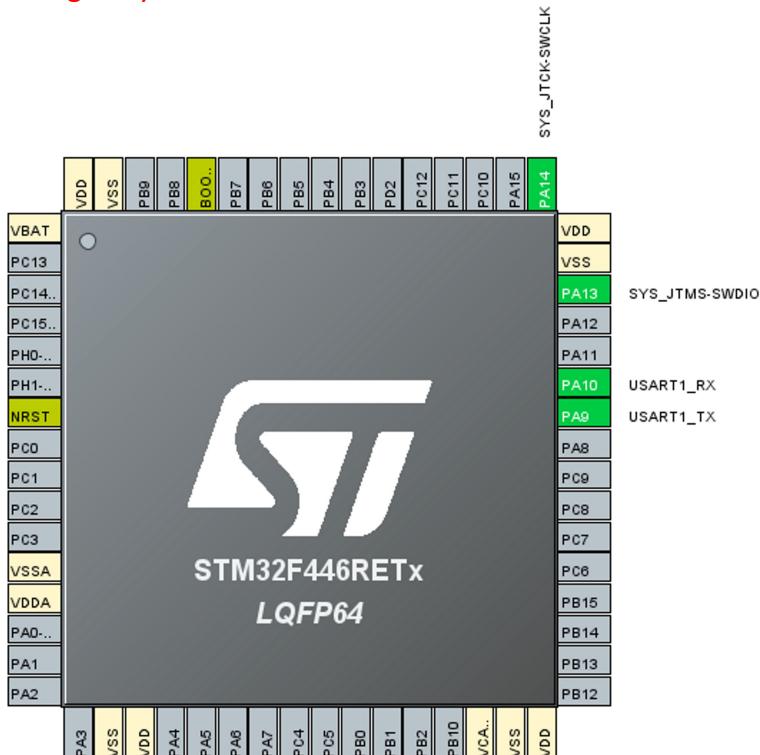
➤ HAL

Teori

- TTL, dijital mantık devrelerinde yaygın olarak kullanılan bir elektriksel sinyal standartıdır. Düşük gerilim seviye olarak 0V ile yüksek gerilim seviyeleri için 3.3V veya 5V kullanır.
- İletişimi USB, RS232 veya RS485 üzerinden sağlamak için TTL dönüştürücü kullanmamız gereklidir. USB için FT232, RS232 için MAX3232 veya MAX232, RS485 için MAX485 gibi entegreleri kullanabiliriz.
- Nucleo kartlarının USART2 pini ile bağlantı yapıldığında, terminal ile haberleşmeyi kartın üzerindeki USB ile sağlayabiliyoruz. USART2 dışındaki diğer USART pinlerini kullanırsak bu pinlerin çıkışına TTL dönüştürücü kullanmamız gereklidir.
- Seri Port işlemlerinde kullandığınız TTL dönüştürücü bilgisayarınızda gözükmüyorsa çipin driverini yüklemeniz gerekiyor. [FT232RL](#) için linkten driver indirip aygit yöneticisinden güncelleyebilirsiniz.
- Printf komutu ile veri göndermek için <https://www.youtube.com/watch?v=SpTh30wTmcM> ile <https://www.youtube.com/watch?v=WnCpPf7u4Xo> linklerindeki videoları inceleyebiliriz.
- Printf fonksiyonu ve türevleri hakkında bilgi almak için <https://bilgisayarkavramlari.com/2012/05/31/printf-sprintf-fprintf/> link üzerinden bakabiliriz.

Polling

Konfigürasyon Kısımları



Pin N...	Signal on Pin	GPIO out...	GPIO mo...	GPIO Pull...	Maximum...	User L...	Modified
PA2	USART2_RX	n/a	Alternate ...	No pull-up...	Very High	<input type="checkbox"/>	
PA3	USART2_TX	n/a	Alternate ...	No pull-up...	Very High	<input type="checkbox"/>	

Mode Asynchronous

- Sadece Transmit yapacağımızdan Data Direction kısmından Transmit Only seçeneğini seçebiliriz.

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

Kod Kısımları

- Sprintf komutu için stdio.h, strlen kütüphanesi için string.h kütüphanesini ekledik.

```
20 /* Includes -----  
21 #include "main.h"  
22 #include "stdio.h"  
23 #include "string.h"  
24 /* Private includes
```

- Göndereceğimiz ifadeleri tutacağımız char değişkenli dizi ekliyoruz.

```
46 /* USER CODE BEGIN PV */  
47 char tx_buff[50];  
48 uint8_t vize=65,final=87;  
49 float ortalama;  
50 /* USER CODE END PV */
```

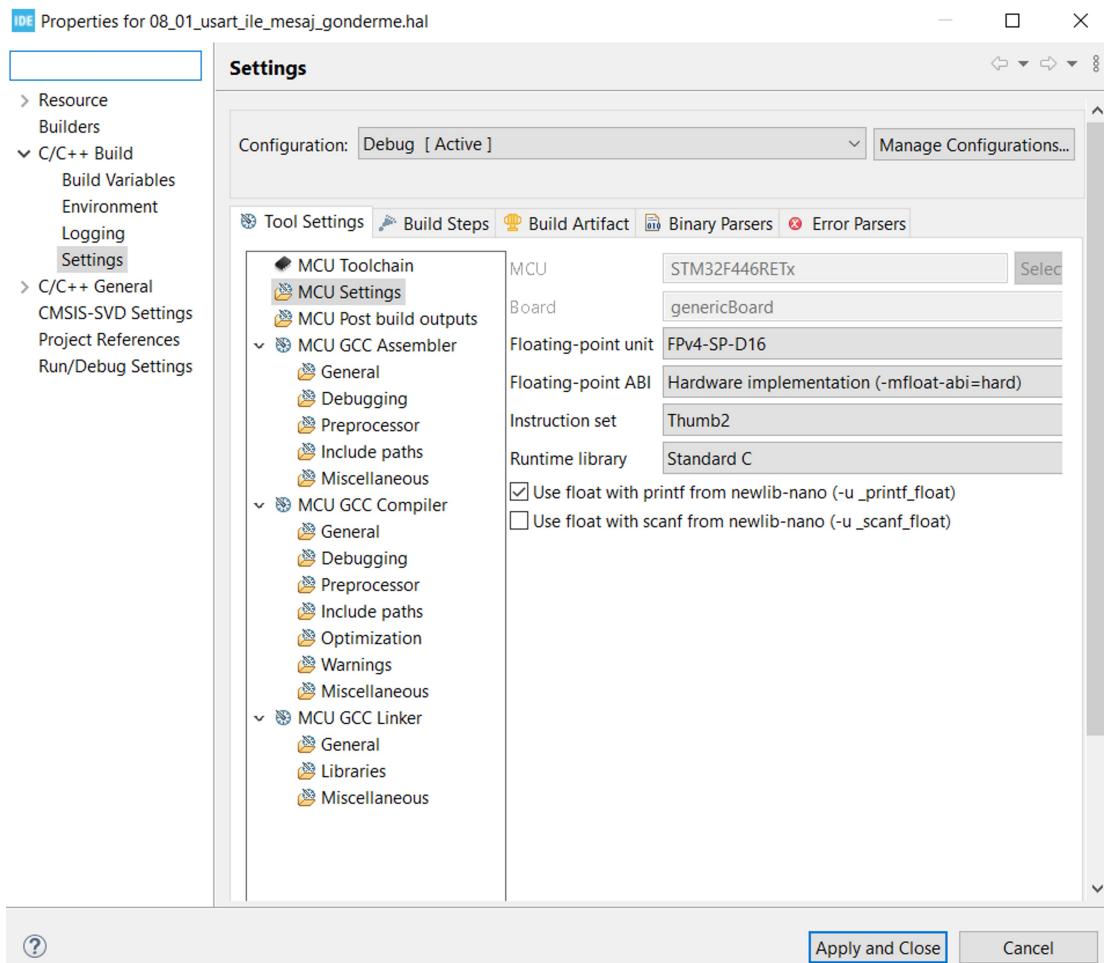
- Sprintf komutu ile yazdırma yaparken fonksiyon içinde de kullanabiliriz ya da ayrı satırda yazıp daha sonra fonksiyon içinde boyutunu belirtmek için strlen fonksiyonunu kullanırız.
- Strlen yerine sizeof fonksiyonu da kullanabilir .
- \n, metinde bir alt satıra geçmek için \r ise metinde paragraf başı yapmak için kullanılır.

```
109 /* USER CODE BEGIN 2 */  
110 ortalama=(vize * 0.4) + (final * 0.6);  
111 sprintf(tx_buff,"Vize:%d, Final:%d, Ortalama:%.2f\r\n",vize,final,ortalama);  
112 /* USER CODE END 2 */
```

- USART işleminde ilk olarak polling mode için HAL_UART_Transmit() fonksiyonunu kullanacağız.
- Fonksiyon için 4 parametre giriyoruz. Birincisi UART ayarlarının tutulduğu veri yapısı, ikincisi gönderilecek veri, üçüncüsü verinin kaç karakter göndereceğimiz yani kaç byte olduğu ve son olarak dördüncüsü veri aktarımının tamamlanması için beklenen maksimum süredir.
- uint8_t tipinde bir data göndermemizi istiyor fakat biz char değişkeni kullandığımızdan bir çevrim yapmamız gerekiyor.

```
102 /* USER CODE BEGIN WHILE */  
103 while (1)  
104 {  
105 /* USER CODE END WHILE */  
106  
107 /* USER CODE BEGIN 3 */  
108 HAL_UART_Transmit(&huart2, (uint8_t *)tx_buff, strlen(tx_buff), 100);  
109 HAL_Delay(500);  
110 }  
111 /* USER CODE END 3 */  
112 }
```

- Float değeri yazdırmak için bir ayar yapılmalıdır. Bunun için proje dosyasına sağ tıklayıp Properties tıklanır. C/C++ Build kısmından Settings kısmına gelinir ve buradan Tool Settings kısmından MCU Settings kısmından -u_printf_float kutucuğu işaretlenir.



- Timeout süresi, milisaniye cinsinden belirtilir. Kod kısmında timeout süresi 50 olarak belirledik. Bu, veri gönderme işleminin en fazla 50 milisaniye sürmesi gerektiği anlamına gelir. Eğer belirtilen süre içinde veri gönderimi tamamlanamazsa, iletişim timeout olur ve ilgili hata durumu işlenebilir.
- Timeout süresini ihtiyaçlarınıza göre ayarlayabiliriz. Daha uzun süreler belirlemek, daha yavaş veya yoğun bir iletişim ortamında güvenilirlik sağlayabilir. Ancak, çok uzun timeout süreleri kullanmak, iletişim hatalarını algılama ve hızlı bir şekilde hata durumlarına yanıt verme yeteneğini azaltabilir.
- Timeout süresini öğrenmek için farklı yollara başvurabiliriz.
Biz Hal_GetTick() fonksiyonunu kullanarak ne kadarlık bir veri gönderme süresi olduğunu buluruz. Bize sonuç verirken 1ms gecikme ekler.
- Kodu aşağıdaki gibi düzenleriz. Int değişken türünde time1 ve time değişkenlerini globalde atarız. Daha sonra fonksiyonun dönüşünden gelen değeri önceki satırda gelen değeri çıkararak sonuca ulaşırız.
- Debug girdiğimizde time değişkenine baktığımızda 36 değeri döndürüyor. Çıkan sonuctan 1 çıkarırız. Sonuç olarak toplamda 35ms'de veriyi gönderiyor. Böylece Timeout için girdiğimiz 50 değeri yeterlidir.
- Hesap yaparak öğrenmek istersek 35 byte veri gönderdiğimizi biliyoruz. Bu değeri bir byte için geçen süre ile çarparsak buluruz.
9600 baud rate için bir byte'in iletiminde geçen süre $1/9600$ den çıkan sonucu 10 ile çarparız daha sonra gönderilen byte'teki adeti olan 35 ile çarparsak sonucunda 35ms olduğunu buluruz.
- Sonuç olarak Polling metodunu kullanarak ana döngüde transmit yaparken 35ms bekleme yapıyor. Bu bekleme döngünün çalışırken başka yapılan bir işi yavaşılatır.

```

105     while (1)
106     {
107         /* USER CODE END WHILE */
108
109         /* USER CODE BEGIN 3 */
110         timel = HAL_GetTick();
111
112         HAL_UART_Transmit(&huart1, (uint8_t *)tx_buff, strlen(tx_buff), 50);
113
114         time = HAL_GetTick() - timel;
115
116         HAL_Delay(500);
117     }
118     /* USER CODE END 3 */
119 }

```

- Printf komutu ile yazdırma istersek aşağıdaki kodu ekleriz. Bu kod ile while döngüsünde Transmit fonksiyonunu yazmamıza gerek kalmıyor.

```

60 /* USER CODE BEGIN 0 */
61 int _write(int file, char *ptr, int len)
62 {
63     HAL_UART_Transmit(&huart2, (uint8_t *)ptr, len, HAL_MAX_DELAY);
64     return len;
65 }
66 /* USER CODE END 0 */

112     while (1)
113     {
114         /* USER CODE END WHILE */
115
116         /* USER CODE BEGIN 3 */
117         //HAL_UART_Transmit (&huart1, tx_buff, strlen(tx_buff), 100);
118         printf("Vize: %d, Final:%d, Ortalama:%.2f\r\n",vize,final,ortalama);
119         HAL_Delay(500);
120     }
121     /* USER CODE END 3 */
122 }

```

Interrupt

Konfigürasyon Kısmı

- Eğer Interrupt kullanacaksak aktif edilmesi gereklidir.

NVIC Interrupt Table	Enabled	Preemption Priority	Sub Priority
USART1 global interrupt	<input checked="" type="checkbox"/>	0	0

Kod Kısmı

- USART işleminde interrupt mode olarak HAL_UART_Transmit_IT() fonksiyonunu kullanacağız. Böylece döngünün ana akışını kesmeden veriyi yollamış olacağız.
Ayrıca HAL_UART_TxCpltCallback, geri çağrıma fonksiyonu ile UART iletişim modülü tarafından bir veri iletim işlemi tamamlandığında otomatik olarak çağırarak yapılması gereken işlemleri gerçekleştirebiliriz.
- Interruptı kullanırken iki türlü kullanabiliriz. Birincisi, Transmit fonksiyonu ana döngü içerisinde yazıp çalıştırabiliriz daha sonra kesmeye girdiğinde Callback fonksiyonu ile başka işlemler gerçekleştirebiliriz.

```

123     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
124 }
125 /* USER CODE END 3 */
126 }

```

- İkincisinde ana döngüye girmeden bir kere Transmit edip daha sonra Callback fonksiyonuyla transfer işlemi gerçekleştirebiliriz.
- Burada döngüden her çıktığında kesmeye gidiyor

```

62 /* USER CODE BEGIN 0 */
63 void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
64 {
65     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
66 }

```

```

110     HAL_UART_Transmit_IT(&huart1, tx_buff, strlen(tx_buff));
111     /* USER CODE END 2 */

```

- Her iki durumda da ana döngü bloke olmamış olacaktır.

DMA

Konfigürasyon Kısmı

- Eğer DMA kullanacaksak aktif edilmesi gereklidir.
- Dairesel modda, DMA verileri iletmeye devam edecektir. Tüm verileri ilettikten sonra, otomatik olarak baştan başlayacaktır.
- Hafızada sadece 1 byte yer kaplayan karakterleri gönderdiğimiz için Data With Byte olarak seçilmiştir.

DMA Request	Stream	Direction	Priority
USART1_TX	DMA2 Stream 7	Memory To Peripheral	Low

Add Delete

DMA Request Settings

Peripheral		Memory	
Mode	Circular	Increment Address	<input type="checkbox"/>
Use Fifo	<input type="checkbox"/>	Threshold	<input type="text"/>
Data Width	Byte	Byte	<input type="text"/>
Burst Size	<input type="text"/>	<input type="text"/>	<input type="text"/>

Kod Kısmı

- USART işleminde DMA mode olarak HAL_UART_Transmit_DMA() fonksiyonunu kullanacağız.
- DMA ayrıca kesme ile aynı şekilde çalışır. Böylece kesme için kullanabildiğimiz Callback fonksiyonlarını kullanabiliriz.

```

131     HAL_UART_Transmit_DMA(&huart1, tx_buff, strlen(tx_buff));
132 }
133 /* USER CODE END 3 */
134 }

```

- Ana döngüye girmeden hem de Callback fonksiyonu çağrımadan da gönderme işlemi yapabiliriz.

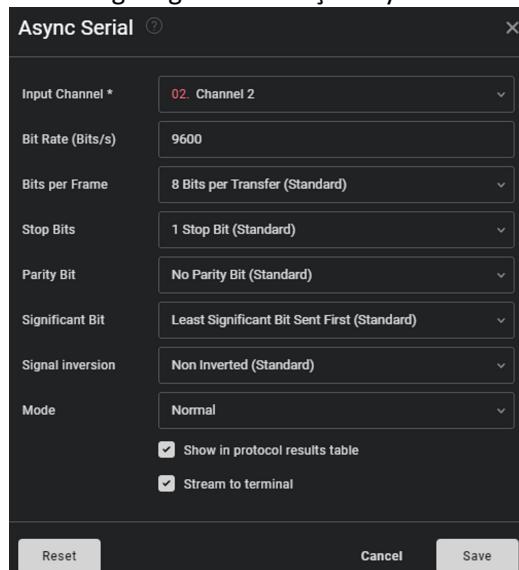
```

115     HAL_UART_Transmit_DMA(&huart1, tx_buff, strlen(tx_buff));
116 /* USER CODE END 2 */

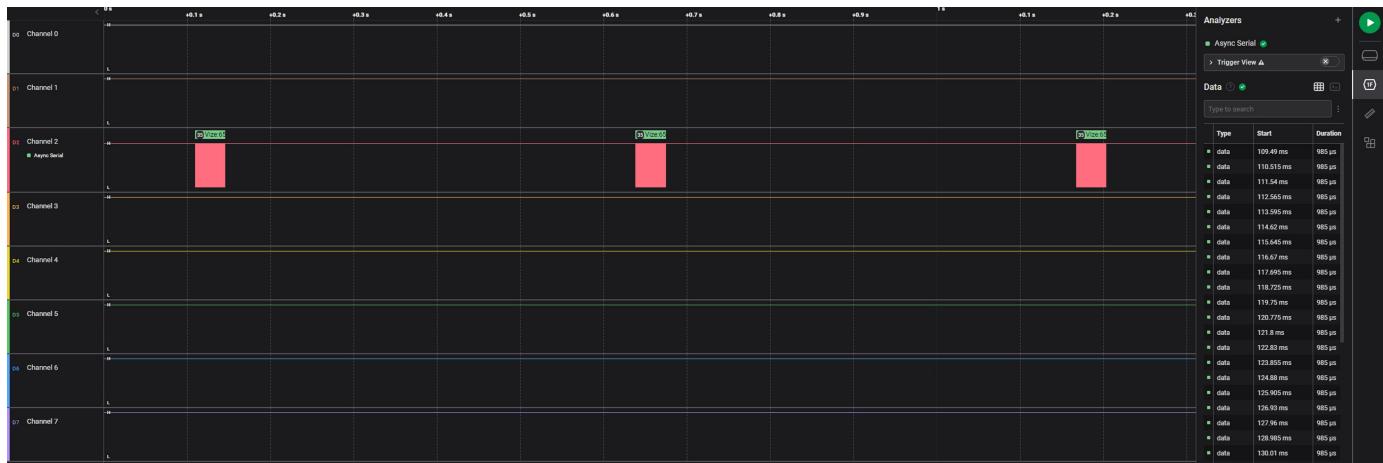
```

Logic Analyzer Kısmı

- Ayrıca seri port dışında logic analyzer ile bakaraka sonuçları elde edebiliriz.
Bağladığımız kanal için Async Serial seçimi yapıp benzer ayarlamaları yapıyoruz.



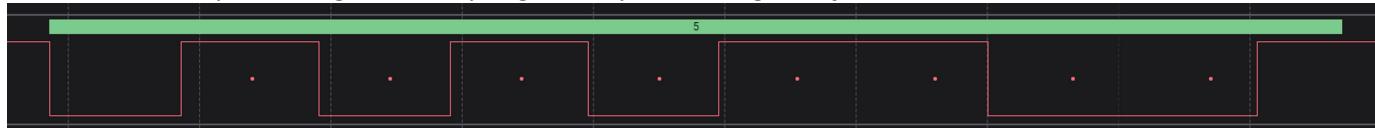
- İşlemcinin Tx ucunu Logic Analyzer'da Channel2'ye bağladık.
- Sonuç olarak 500ms aralıklarla 35 byte veriyi 35ms'de yolluyor.
- Ayrıca veri göndermeyi 50 ms altında yaparak işlemci zaman aşımına uğramıyor. Eğer kod kısmında 50 değil de 35'in altında bir değer yazsaydık veri kaybı olacaktı.



- Verileri görüntülerken gelendataları ASCII olarak görüntülüyoruz.

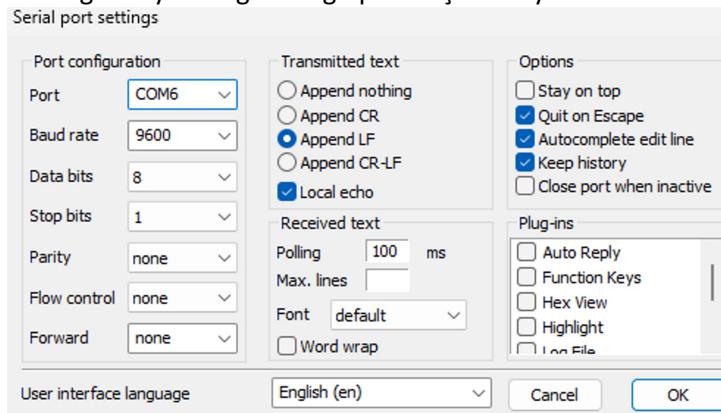


- Gelen verideki 5 verisine detaylı bakalım. 5'i ASCII'den Binary'e çevirmemiz gerekiyor.
- Ekrandan de değiştirebiliriz ya da <https://www.rapidtables.com/convert/number/ascii-to-binary.html> linkten bu işlemi yapabiliriz.
- Ayrıca https://www.netdunyasi.com/blog/uploads/images/202105/image_750x_60985f549fe84.jpg linkten ASCII tablosuna bakabiliriz.
- Binary'e çevirdiğimizde 00110101 değerini görürüz.
- İlk ve son bitler start ve stop bit olmak üzere nokta ile belirlmiş 8 bitlik data ile birlikte toplamda 10 bit var.
- Burada sırayla bakıldığından binary değeri ile aynı sonucu görmüş oluruz.

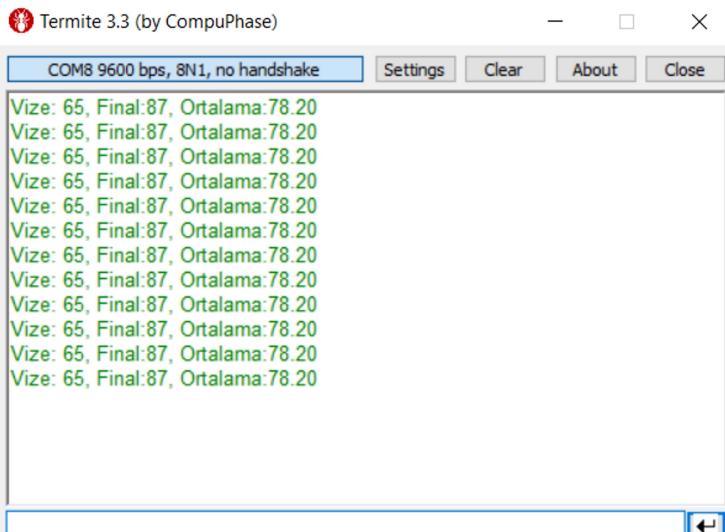


Seri Port Kısı

- Seri port yazdırma için Termite programını kullanıyoruz. Programı açtıktan sonra bazı ayarları yapmamız gerekiyor. Bağlı olduğu port seçimi ile yukarıda belirlediğimiz 9600 baud rate değerini seçiyoruz.



- Port ayarlarını yaptıktan sonuçları görüntüleyebiliriz.



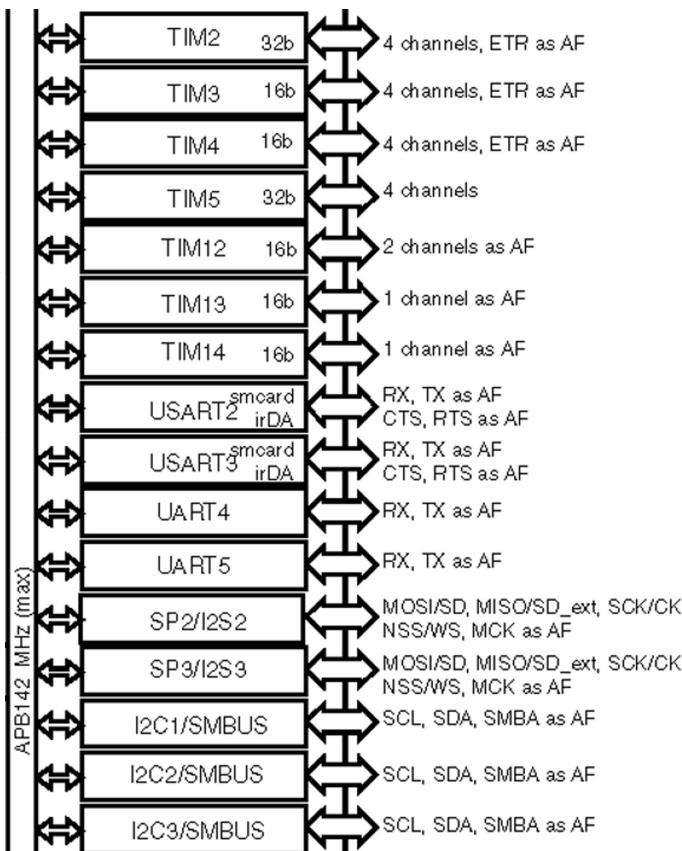
➤ REGISTER

Konfigürasyon Kısımlı

- RCC için yazdığımız fonksiyon aşağıdaki gibidir.

```
3④ 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 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	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	

18.biti 1 yapıyoruz.

Bit 18 **USART3EN**: USART3 clock enable

Set and cleared by software.

0: USART3 clock disabled

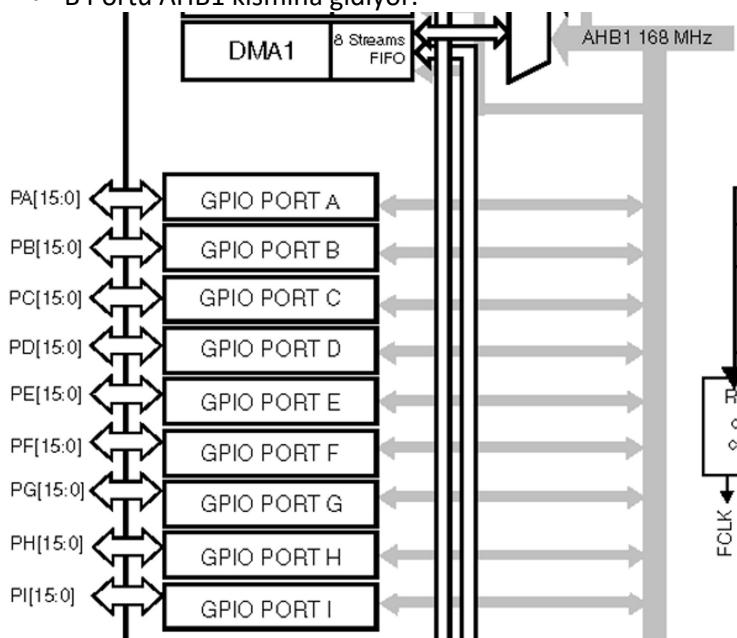
1: USART3 clock enabled

RCC->APB1ENR |= 1 << 18;

- USART3 hangi porta bağlı olduğunu öğrenmek için datasheet'e bakarız. B portun 10. ve 11.pinine 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/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_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	-	-

- B Portu AHB1 kismına 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	GPIOF E N	GPIOEEN	GPIOD EN	GPIOC EN	GPIOBEN	GPIOAEN		
		rw				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							
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							

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 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

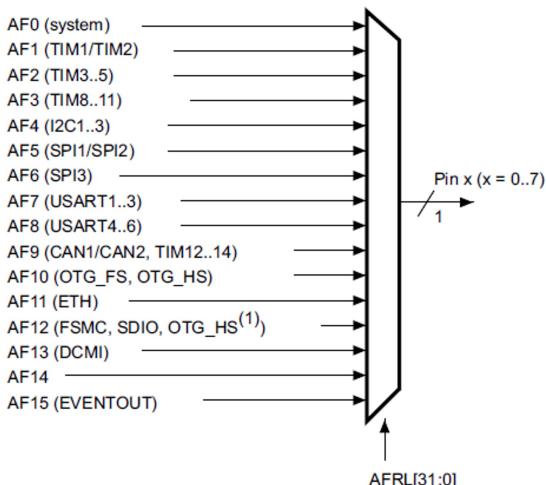
AFRH_y selection:

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

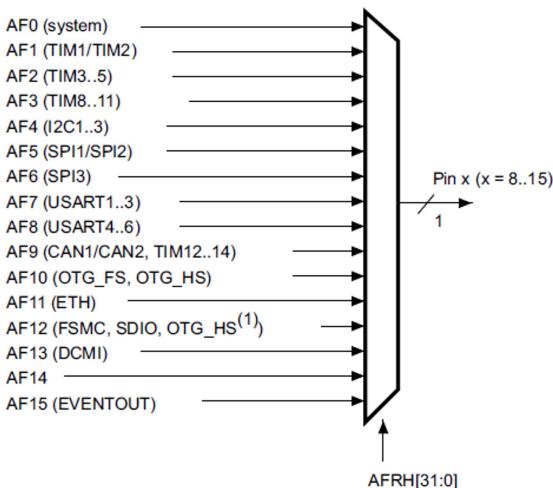
- Seçtiğimiz USART3 çevresel birimi hangi AF'de olduğunu bilmek için Reference Manuel kitapçığına bakıyoruz.

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



- USART3, AF7'de bulunuyor. Böylece pinlere AF7 için olan 0111 tanımlaması yapacağız.
- AFR'nin High ve Low olduğunda belirtmemiz gerekiyor. High kullandığımızdan dizin kısmına 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 hesabı için aşağıdaki formül kullanılır. USARTDIV ile istenen baud rate değeri hesaplanır ve bulunan değer BRR register'a yazılır.

$$\text{Tx/Rx baud} = \frac{f_{CK}}{8 \times (2 - \text{OVER8}) \times \text{USARTDIV}}$$

- OVER8 için CR1 registerinde oversampling by 16 kullandığımızdan bu değeri 0 alıyoruz.
- Baud rate ayarını 9600 yapmak için USARTDIV 273,4375 bulunur.
- Hesap yapmak yerine Reference Manual'deki tablolardan da bakılabilir.
- Biz 3.satırda değeri bulduk.

Oversampling by 16 (OVER8=0)							
Baud rate		$f_{PCLK} = 42 \text{ MHz}$			$f_{PCLK} = 84 \text{ MHz}$		
S.No	Desired	Actual	Value programmed in the baud rate register	% Error = (Calculated - Desired)B.Rate /Desired B.Rate	Actual	Value programmed in the baud rate register	% Error
1.	1.2 KBps	1.2 KBps	2187.5	0	1.2 KBps	NA	0
2.	2.4 KBps	2.4 KBps	1093.75	0	2.4 KBps	2187.5	0
3.	9.6 KBps	9.6 KBps	273.4375	0	9.6 KBps	546.875	0
4.	19.2 KBps	19.195 KBps	136.75	0.02	19.2 KBps	273.4375	0
5.	38.4 KBps	38.391 KBps	68.375	0.02	38.391 KBps	136.75	0.02
6.	57.6 KBps	57.613 KBps	45.5625	0.02	57.613 KBps	91.125	0.02
7.	115.2 KBps	115.068 KBps	22.8125	0.11	115.226 KBps	45.5625	0.02
8.	230.4 KBps	230.769 KBps	11.375	0.16	230.137 KBps	22.8125	0.11
9.	460.8 KBps	461.538 KBps	5.6875	0.16	461.538 KBps	11.375	0.16
10.	921.6 KBps	913.043 KBps	2.875	0.93	923.076 KBps	5.6875	0.93
11.	1.792 MBps	1.826 MBps	1.4375	1.9	1.787 MBps	2.9375	0.27
12.	1.8432 MBps	1.826 MBps	1.4375	0.93	1.826 MBps	2.875	0.93
13.	3.584 MBps	N.A	N.A	N.A	3.652 MBps	1.4375	1.9
14.	3.6864 MBps	N.A	N.A	N.A	3.652 MBps	1.4375	0.93

- Ondalıklı sayının tam kısmına mantissa, küsürat kısmına fraction denir.
- Buna göre mantissa değeri 273 yani 0x111 bulunur. Geriye kalan 0.4375 değerini fraction değerine dönüştürmek için OVER8 değeri 0 olduğundan 16 ile çarpılır sonucunda 7 yani 0x7 bulunur.
- Böylece BRR registerına 0x1117 değeri yazılır.

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.

`USART3->BRR |= 0x1117;`

- USART_BRR değerini döndürecek fonksiyon ile de yazabiliriz.

```

int BRR(double baud)
{
    double decimal = (42*1000000) / (baud*16);

    int Mantissa = (int)decimal;
    int Fraction = ceil((decimal-Mantissa)*16);

    return ((Mantissa << 4) + Fraction);
}

```

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 **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);
```

- Üsteki uyaridan sebep UE regitri 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															
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

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.

```
void USART_Config(void)
{
    RCC->APB1ENR |= 1 << 18; //USART3EN

    USART3->BRR |= 0x1117; //BaudRate 9600
    USART3->CR1 |= (1 << 2); //Receiver Enable
    USART3->CR1 |= (1 << 3); //Transmitter Enable
    USART3->CR1 |= (1 << 5); //RXNE Interrupt Enable
    USART3->CR1 |= (0 << 10); //Parity Control Disabled
    USART3->CR1 |= (0 << 12); //Word Length 8 Data Bits
    USART3->CR2 |= (0 << 12); //1 Stop bit
    USART3->CR1 |= (1 << 13); //USART Enable
}
```

Kod Kısmı

- Şimdi interrupt için fonksiyon yazacağız. 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
Reserved						CTS	LBD	TXE	TC	RXNE	IDLE	ORE	NF	FE	PE
						rc_w0	rc_w0	r	rc_w0	rc_w0	r	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);
```

```
49 void NVIC_Config(void)
50 {
51     NVIC->ISER[1] |= (1 << 7);           //Interrupt Set Enable Register
52 }
```

- Şimdi Interrupt fonksiyonu yazıyoruz. Öncelikle değişken ataması yapıyoruz. Haberleşme durumunu Status Register ile bu değişkene yazacağız.
- Daha sonra 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 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;
```

```
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 yani her dizideki elemanlarını alıyoruz ve her elemanı Send_Char'a gönderiyoruz ve burada Data register'a yazarak karşı tarafa mesaj ulaşıyor