

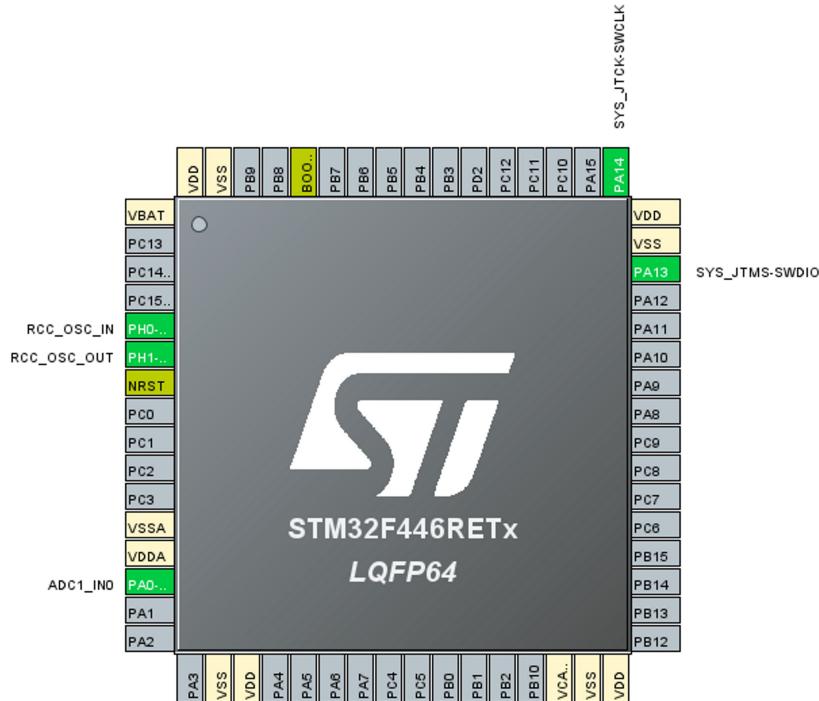
ADC Verisi Okuma

25 Aralık 2021 Cumartesi 00:56

ADC Verisi Okuma

➤ HAL

Konfigürasyon Kısımları



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

- Analag 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

- 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 |
| End Of Conversion Selection | EOC flag at the end of single channel convers |
- 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 */
62void 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 */

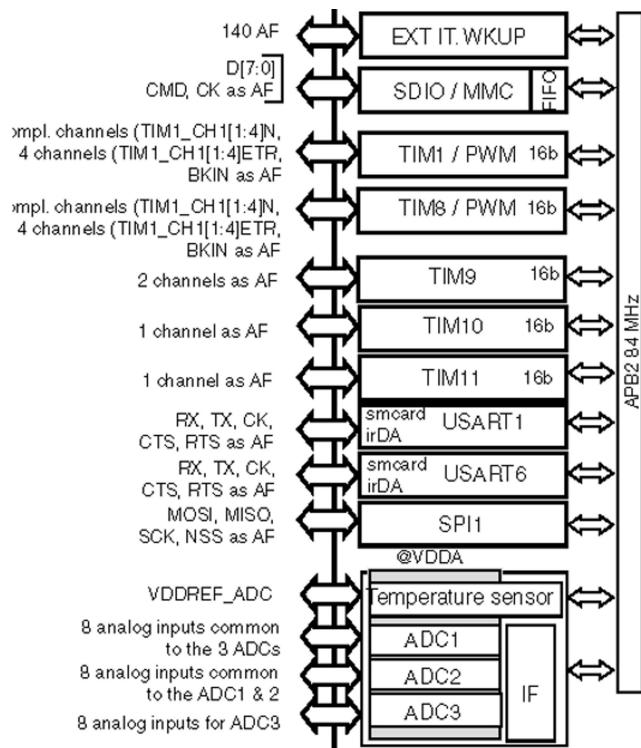
```

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

➤ 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												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 EN	Reser- ved	SPI1 EN	SDIO EN	ADC3 EN	ADC2 EN	ADC1 EN	Reserved	USART 6 EN	USART 1 EN	Reserved	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

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

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

This bit is set and cleared by software.

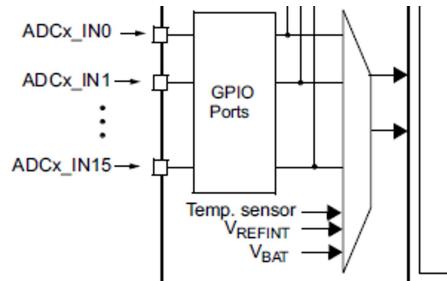
Note: 0: Disable ADC conversion and go to power down mode

1: Enable ADC

- ADC aktif etmek için ilk bit 1 yapıldı.

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

- ADC_SPMR 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

ADC sample time register 2 (ADC_SMPR2)

Address offset: 0x10

Reset value: 0x0000 0000

- 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	
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	rw	rw	rw	rw

Bits 17:16 **ADCPRE**: ADC prescalers

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;          //PLLCSR
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 }

```

`void ADC_Config(void)`

```

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.

```
3 uint8_t adc_value;
```

- 8 bitlik dönüşüm değeri olan 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 }
```

```
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	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.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.

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

- Çevrimin bitip bitmediğini ADC_SR registerinden öğreniyoruz.

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	rc_w0	rc_w0	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]															
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](#).

- 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