ADC with stm32 with multiple channels

# ADC with dma

## Active clock (RCC configurations)

metin, ekran görüntüsü, yazılım, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

## Clock configurations

metin, diyagram, sayı, numara, yazılım içeren bir resim

Açıklama otomatik olarak oluşturuldu

## ADC1 parameter settings

-Activate IN1 and IN2.

-Clock Prescaler : The ADC work at 36 MHZ as maximum frequency , the clock line of adc id ---APB2 with frequency 84 MHZ, so we divide PCLK2 by 4.

-Scan conversion Mode : Enabled for multi ADC reading.

-Continuos conversion mode: Enabled.

-DMA Continuos request: Enabled.

-Number of conversion: 2 because we use 2 channels.

-Press Rank and choose channels required.

Note: ADC is so fast, so that case fast DMA interrupt and the code wouldn’t work so you have to make it slowest, so increase the simple time and decrease clock prescaler.

metin, ekran görüntüsü, yazılım, web sayfası içeren bir resim

Açıklama otomatik olarak oluşturuldu

## ADC DMA settings

-Press add. Select ADC1. You can choose stream 0. Choose priority u want , we will use very high. Choose circular mode .

-Increment address is choosen if address change, address not change in peripheral but change at memory so we will choose only memory.

-Data Width, Byte is 8, half word 16 and word is 32 bit, because ADC is 12 bit , we can choose Half word it is okay for us.

metin, yazılım, web sayfası, bilgisayar simgesi içeren bir resim

Açıklama otomatik olarak oluşturuldu

Now generate the code, and let’s start writing code.

## Code writing

Lets define three variables in our code. The first is an array it’s the dma buffer, to the second and the third varibles we will write the data form the buffer.

|  |
| --- |
| /\* USER CODE BEGIN PV \*/  uint16\_t adc\_value[2];  uint16\_t adc1\_ch0,adc1\_ch1;  /\* USER CODE END PV \*/ |

Refer to stm32f4xx\_hal\_adc.c we can learn how to read adc values with dma.

|  |
| --- |
| \*\*\* DMA mode IO operation \*\*\*  ==============================  [..]  (+) Start the ADC peripheral using HAL\_ADC\_Start\_DMA(), at this stage the user specify the length  of data to be transferred at each end of conversion  (+) At The end of data transfer by HAL\_ADC\_ConvCpltCallback() function is executed and user can  add his own code by customization of function pointer HAL\_ADC\_ConvCpltCallback  (+) In case of transfer Error, HAL\_ADC\_ErrorCallback() function is executed and user can  add his own code by customization of function pointer HAL\_ADC\_ErrorCallback  (+) Stop the ADC peripheral using HAL\_ADC\_Stop\_DMA() |

-Start ADC DMA after init, don’t forget to cast buffer to 32 bit .

|  |
| --- |
| /\* USER CODE BEGIN 2 \*/  HAL\_ADC\_Start\_DMA(&hadc1, (uint32\_t\*)adc\_value, 2);  /\* USER CODE END 2 \*/ |

* In while loop let’s write data from the adc\_value buffer, to the variables.

|  |
| --- |
| **while** (1)  {  /\* USER CODE END WHILE \*/  /\* USER CODE BEGIN 3 \*/  adc1\_ch0 = adc\_value[0];  adc1\_ch1= adc\_value[0];  } |