By Jacoby Roy

ADC with DMA using ping-pong buffers on STM32F446

This demo shows how to configure the ADC with the DMA to collect data asynchronously to the CPU.

Other great resources:

* https://www.digikey.ca/en/maker/projects/getting-started-with-stm32-working-with-adc-and-dma/f5009db3a3ed4370acaf545a3370c30c

## Tools:

* Nucleo STM32F446ZE
* STM32CubeIDE

## I/O Selection

For this guide, we will use the analog pin PA3, which connects to channel IN3 of the ADCs.

From UM1974 p.37:

Une image contenant texte, nombre, capture d’écran, Police

Description générée automatiquement

In Cube, I select ADC1 and set the channel IN3:

Une image contenant texte, capture d’écran, logiciel, Page web

Description générée automatiquement

I confirm that the pin PA3 is selected:

Une image contenant texte, capture d’écran, Logiciel de graphisme, Logiciel multimédia

Description générée automatiquement

## Clock configuration

From RM0390 p.361, we find that the ADC is in the APB2 clock domain.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

In the clock tree, we find APB2 to find the configured clock frequency (84MHz in our case) for the PCLK2 clock used by the ADC.

NOTE: you can set it to a different value and adjust the calculations for your clock frequency

Une image contenant texte, capture d’écran, diagramme, nombre

Description générée automatiquement

By looking at p.369, we find that the conversion minimum conversion time for given resolutions:

Une image contenant texte, capture d’écran, Police, algèbre

Description générée automatiquement

Since we want max resolution (12 bits), it means we need at least 15 clock cycles of the ADC clock as a sampling time.

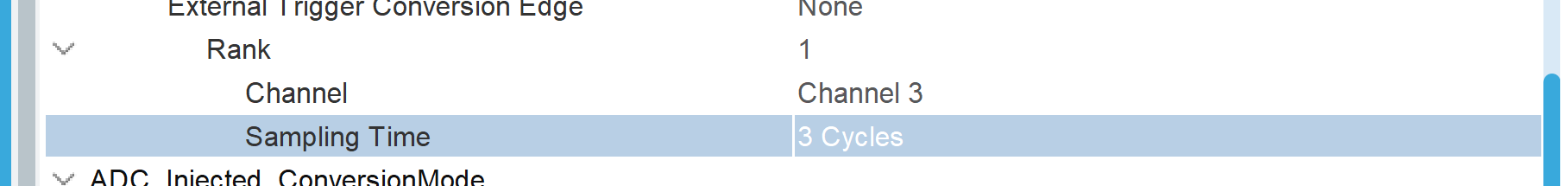
In our case, when trying to set the prescaler in Cube for the minimum value, the smallest we have is 4, meaning the ADCCLK runs at PCLK2/4 = 84MHz/4 = 21MHz.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

This gives a minimum sampling time of 15 clock cycles \* 1/21MHz = 0.7 us

You can play with the sampling time either by adjusting the clock or by adjusting the sampling time in the channel configuration:



## Conversion mode and DMA

Some ADCs of other manufacturers are able to sample values of the same channel and each sample into different registers. However, this chip’s ADC stores a given channel sample into only one register, therefore we cannot use the SCAN mode for only one channel. BUT it is still possible to use SCAN mode for a group of channel if necessary.

However, we can still use the CONTINUOUS mode, which requests samples from the ADC continuously. Combining this with DMA allows fetching data in the register as it is sampled and storing it into memory for further processing, all of this without (or almost without) no CPU involvement.

Be careful with the overrun error has when triggered, the DMA cannot continue fetching data from the ADC. You simply need to clear the interrupt when it is triggered to continue the operation (see p.370).

To configure the ADC to support a continuous polling from DMA, enable the following parameters:

First, go to the ADC DMA configuration window and add a DMA channel in Circular mode. The DMA circular mode simply means that once it completed the number of requests to be transferred at the target address (for example, 4 requests of 4 bytes each at address 0x00 means it will reach address 0xF0 once the 4 requests are completed), it loops back to the base address of transfer (in this example this means it loops back to 0x00 and continuous storing data).

Here we selected Stream0 of DMA2.

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

After this in the parameters settings, enable the “Continuous Conversion Mode” and also enable the “DMA continuous requests” option to allow the DMA to keep the ADC sampling continuously.

Une image contenant texte, capture d’écran, logiciel, Icône d’ordinateur

Description générée automatiquement

You can now generate the code.

## The code

The important functions of the HAL here are these:

* **void** **HAL\_ADC\_ConvHalfCpltCallback**(ADC\_HandleTypeDef\* hadc)

This is the callback function which is called whenever the DMA has performed half of the transactions (here it means fetched half of the samples requested to the ADC). It needs to be defined in your code, so we will put it in main.c

* **void** **HAL\_ADC\_ConvCpltCallback**(ADC\_HandleTypeDef\* hadc)

This callback function is just like the previous one, but is called when all the conversions are done. It also needs to be defined in your code.

* HAL\_StatusTypeDef **HAL\_ADC\_Start\_DMA**(ADC\_HandleTypeDef \*hadc, uint32\_t \*pData, uint32\_t Length)

This function starts the ADC with DMA. The peripheral then runs in the background and you can interact with it through the callbacks

* HAL\_StatusTypeDef **HAL\_ADC\_Stop\_DMA**(ADC\_HandleTypeDef \*hadc)

This function is used to stop the ADC-DMA operations if necessary.

Now, the data will be stored in a buffer of say 512 bytes. When the DMA will have filled up the first 256 bytes, the half complete callback will be called. When it will have filled up the next 256 bytes, the transfer complete callback will be called and then the DMA will loop around and start filling the buffer from the start again.

The callbacks functions must be very short if possible (good practice to keep interrupt context as short as possible), therefore they will simply set flags that will be polled in the main loop. These flags will indicate whether data is ready or not. When data is ready, you can do whatever you want with it (for example, save to the SD card or copy to another buffer which is then saved to the SD card when full).

The important part here is that the operations you are doing with the buffer data are faster than the ADC-DMA, otherwise the data will get overwritten as you process it.

To check the ADC-DMA for the clock and sampling time set, you can turn ON a GPIO in the half-callback and turn it back OFF in the other callback. You can check the period using an oscilloscope.

Good luck!