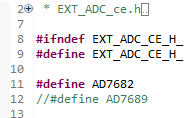
EXT\_ADC Library Manual

If you are not interested in the details about how everything is arranged here, then it is better to refer immediately to the last section of the manual, which is called “How to use”.

This library is a microcontroller driver of the STM32F4 family for data exchange with AD7682 / 7689 ADC. To choose between AD7682 or AD7689, you need to use the #define directive in the file "EXT\_ADC\_ce.h":



The library structure is shown in the following figure, along with libraries that are included using the #include directive for implementing circular buffers (**circular\_buffers\_lib**), filtering measured data (**data\_processing\_lib**), and SPI interface settings:



Figure 1 - File structure

Before continuing to review the library in more detail, you need to read the time diagrams from the AD7682 / 7689 ADC documentation:

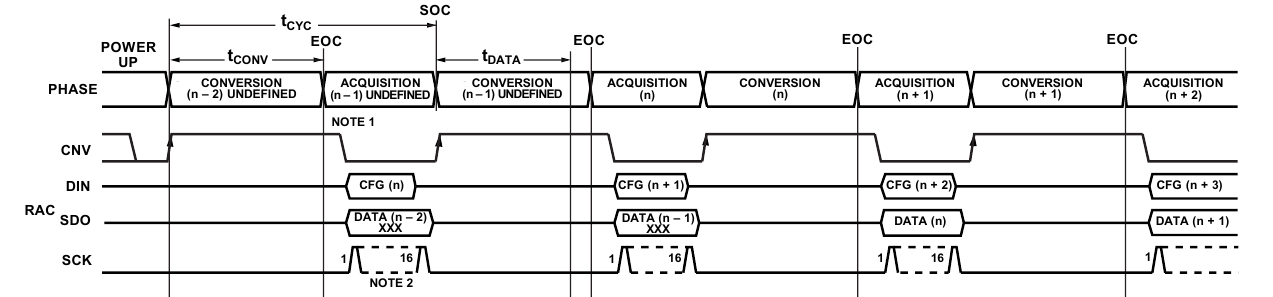


Figure 2 – Timing diagrams without a busy indicator

The **conversion** is triggered by the rising edge of the CNV line, the conversion is followed by the data **acquisition** phase, this is the time interval when receiving and transmitting via SPI occurs. As can be seen from the diagrams, the data to be sent is the value to be written to the Configuration Register, one of the fields of this register and determines from which input channel the measurement will occur.

Also, as can be seen from the diagrams, the data is provided with a shift of 2, since after switching on the power, the value of the Configuration Register register is not defined and requires two transactions before it becomes one.

There are two transmission modes: with and without a busy indicator. The transfer mode with the readiness indicator is when the ADC sets the SDO line to zero when the conversion is completed. In this mode, before starting a new conversion, an additional 17th clock pulse is needed; this is necessary in order for the ADC to transfer the SDO line to the Z-state.

In order for the ADC to work in the mode with busy indicator, it is necessary that at the moment of the conversion end, the CNV line is set to zero (Figure 3).

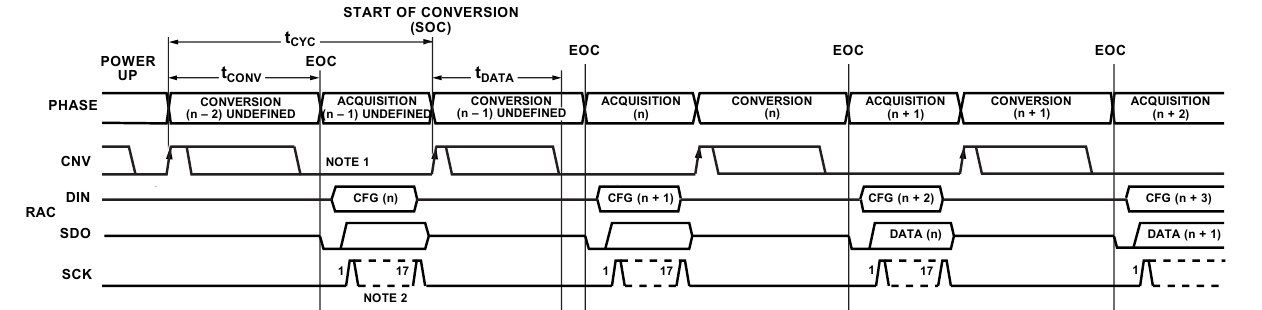


Figure 3 – Timing diagrams with a busy indicator

The **EXT\_ADC\_vMain()** function is an state machine and it is used to control the transfer algorithm. In the file "EXT\_ADC\_ce.h" you need to select the transfer mode using the directive #define:



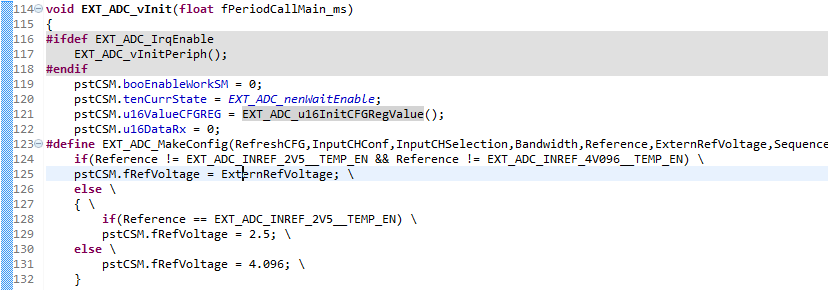
Depending on which of these modes is selected using the #ifdef directive, there are two variants of the state machine **EXT\_ADC\_vMain():**

Figure 4 – State machine for transmission **without** busy indicator

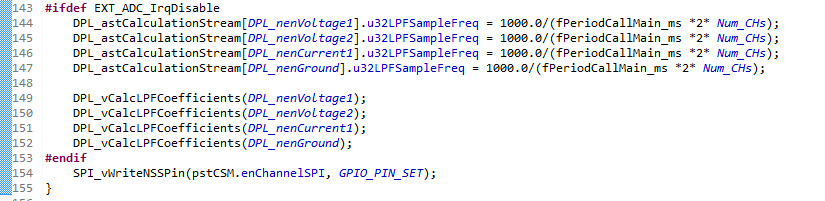
 Figure 5 – State machine for transmission **with** busy indicator

The mode without busy indicator is simpler because there is no need to form an additional clock pulse. Entry into the state machine determines the one of the ADC input channel measurement frequency. The low-pass filter coefficients from the data\_processing\_lib depend on this frequency; therefore, the initialization function takes as an argument the period of entering the state machine in milliseconds for to calculate the low-pass filter coefficients.

Automatic recalculation is not provided for the mode with the busy indicator, therefore, in this case, you must set the correct sampling rate (LPFilterSampleFreq) manually in the file “EXT\_ADC\_ce.h”.



…



The mode with busy indicator makes sense to use for fast transmission between the microcontroller and the ADC, since in this case the state machine triggered on the interrupt after the end of the transmission via SPI and on the readiness indicator. In this case, the **EXT\_ADC\_vMain()** function can be called without any delay, in the infinite loop (while (1) ) of the **main()** functions. The maximum transfer rate in this case is limited only by the time parameters of the ADC.

This project uses a mode without a busy indicator because the OS has tasks with a minimum period of 1 ms. The maximum transfer rate in this case is limited by the period of entry into the state machine.

The state machine is written in such a way that for each interrupt after the end of the transmission via SPI, data is saved to buffers and filtered. The state machine links data exchange with ADC and data processing using **circular\_buffers\_lib** and **data\_processing\_lib**.

These two libraries are written so that they can be easily linked with each other, but at the same time, this does not prevent them from acting as libraries completely independent of each other. The following figure shows in more detail the connection between libraries:

Figure 6 – Relationship between libraries

The libraries **circular\_buffers\_lib** and **data\_processing\_lib** are linked using the function **DPL\_vFillSampleArray()**. This function belongs to the library **data\_processing\_lib**, but inside it is a call **BUF\_vTakeNLastValues ()** function, which is owned by **circular\_buffers\_lib**. The function **BUF\_vTakeNLastValues (BUF\_tenBuffers enBuf, uint8\_t u8NumLastValues, float \* pfRecipientAddr)** takes as the arguments the number of the buffer, the number of elements that need to be taken and the pointer to the array. Using this data, the function simply copies the last values added to the buffer to the array, then this data array is used for the data filtering process.

In the **DPL\_vFillSampleArray()** function, the NoiseReduction also occurs after copying data to the sample array. Subsequent filtering is performed by the **DPL\_vFilterOperation()** function.

How the low pass filter is arranged can be found in this article: http://www.labbookpages.co.uk/audio/firWindowing.html

The way the setup of the filtering process is arranged is shown in the following figure:





Figure 7 – Data processing library organization

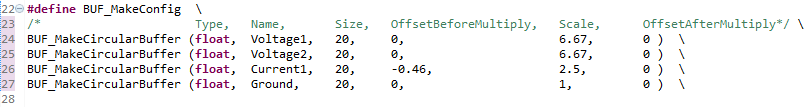
Setup in the file "EXT\_ADC\_ce.h"

1) SPI is configured separately in the file "SPI1\_ce.h", and in order to associate the desired SPI with the EXT\_ADC library, you need to specify its name, port and CLK pin in the file "EXT\_ADC\_ce.h". The last thing is necessary for the formation of the 17th clocking pulse.

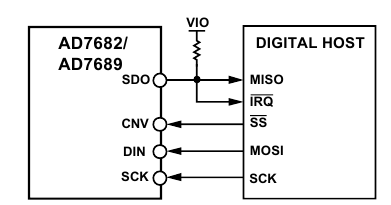


2) The following macro concerns the setting of the Circular buffers library. Buffer creation is configured here. You need to specify the type, name, size, and coefficients for the offset and multiplication. Offset and multiplication occurs before adding a new element to the buffer (function **BUF\_vAddValueInBuffer**()), according to the following formula:

Output = (Input + OffsetBeforeMultiply)\*Scale + OffsetAfterMultiply



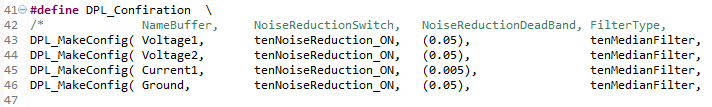
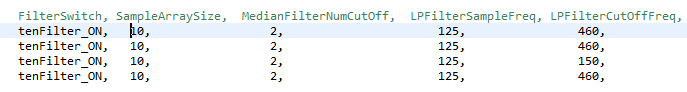
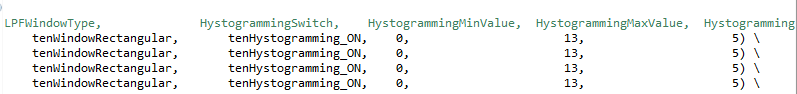
3) The following macro sets the initial values for the fields of the Configuration Register register, defines the initial value of the reference voltage for the ADC and the port and the number of the pin that monitors the busy indicator.



4) Configure the Data processing library. In order for the Data processing library and Circular buffers library libraries to work together, it is important to specify the buffer name correctly here.

The way the filtering process setup works is shown in Figure 7.

How to use

1) Make all settings in the file "EXT\_ADC\_ce.h";

2) Call the EXT\_ADC\_vInit(float fPeriod\_ms) function and pass to it the period for calling the state machine EXT\_ADC\_vMain () in milliseconds;

3) Periodically call the EXT\_ADC\_vMain() function;

4) For to get output values use functions EXT\_ADC\_fGetVCC1(), EXT\_ADC\_fGetVCC2(), EXT\_ADC\_fGetCurrent(), EXT\_ADC\_fGetGND().

IMPORTANT: The library is not designed for data exchange with more than one ADC AD7682/7689!

Example

**int** **main**(**void**)

{

HAL\_Init();

SystemClock\_Config();

\_\_HAL\_RCC\_TIM10\_CLK\_ENABLE();

\_\_HAL\_RCC\_TIM3\_CLK\_ENABLE();

timer\_init(TIM10,*TIM1\_UP\_TIM10\_IRQn*, 5000 , *US*);

timer\_init(TIM3,*TIM3\_IRQn*, 1000 , *US*);

init\_button\_context\_struct(&user\_button);

SPI\_vInit();

EXT\_ADC\_vInit(1);

EXT\_ADC\_vEnableWorkSM();

**while**(1)

{

SPI\_vMain();

**if**(check\_vMain == 1)

{

check\_vMain = 0;

EXT\_ADC\_vMain();

}

**if**(user\_button.flag\_check\_button == 1)

{

user\_button.flag\_check\_button = 0;

user\_button.flag\_button = update\_pin\_state(&user\_button);

}

**if**(user\_button.flag\_button == 1)

{

user\_button.flag\_button = 0;

**printf**("vcc1 = %f\r\n", EXT\_ADC\_fGetVCC1());

**printf**("vcc2 = %f\r\n", EXT\_ADC\_fGetVCC2());

**printf**("current = %f\r\n", EXT\_ADC\_fGetCurrent());

**printf**("gnd = %f\r\n", EXT\_ADC\_fGetGND());

}

}

}

**void** **TIM3\_IRQHandler**(**void**)

{

TIM\_HandleTypeDef htim;

htim.Instance = TIM3;

**if**(\_\_HAL\_TIM\_GET\_FLAG(&htim, TIM\_FLAG\_UPDATE) != *RESET*)

{

**if**(\_\_HAL\_TIM\_GET\_IT\_SOURCE(&htim, TIM\_IT\_UPDATE) !=*RESET*)

{

\_\_HAL\_TIM\_CLEAR\_IT(&htim, TIM\_IT\_UPDATE);

check\_vMain = 1;

}

}

}

**void** **TIM1\_UP\_TIM10\_IRQHandler**(**void**)

{

TIM\_HandleTypeDef htim;

htim.Instance = TIM10;

**if**(\_\_HAL\_TIM\_GET\_FLAG(&htim, TIM\_FLAG\_UPDATE) != *RESET*)

{

**if**(\_\_HAL\_TIM\_GET\_IT\_SOURCE(&htim, TIM\_IT\_UPDATE) !=*RESET*)

{

\_\_HAL\_TIM\_CLEAR\_IT(&htim, TIM\_IT\_UPDATE);

user\_button.flag\_check\_button = 1;

}

}

}

Filter results

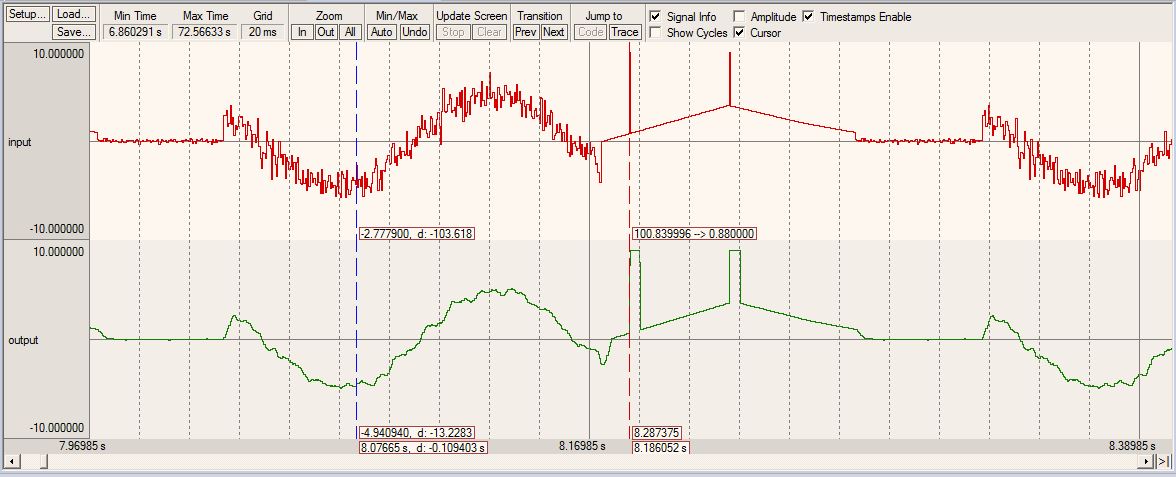


Figure 8 – LPF operation example

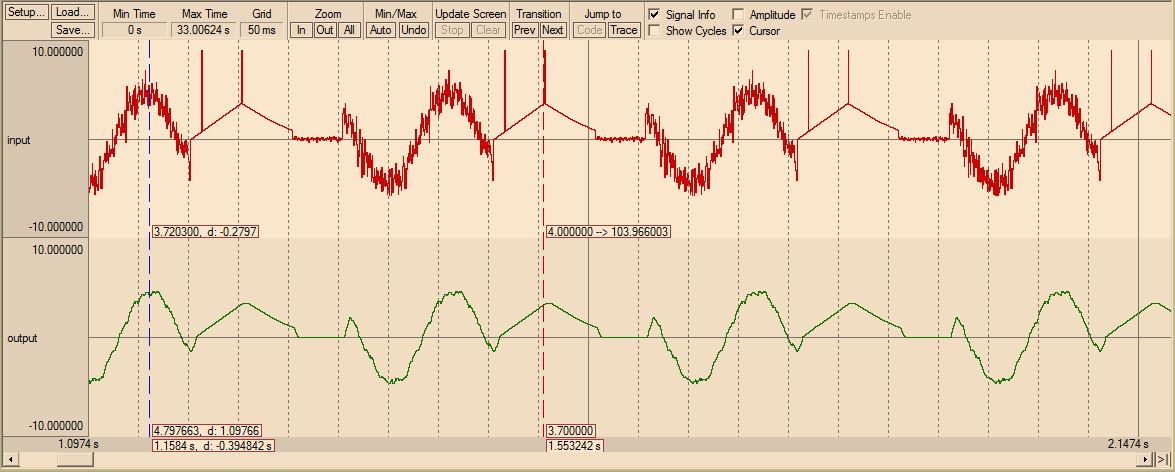


Figure 9 – Median filter operation example