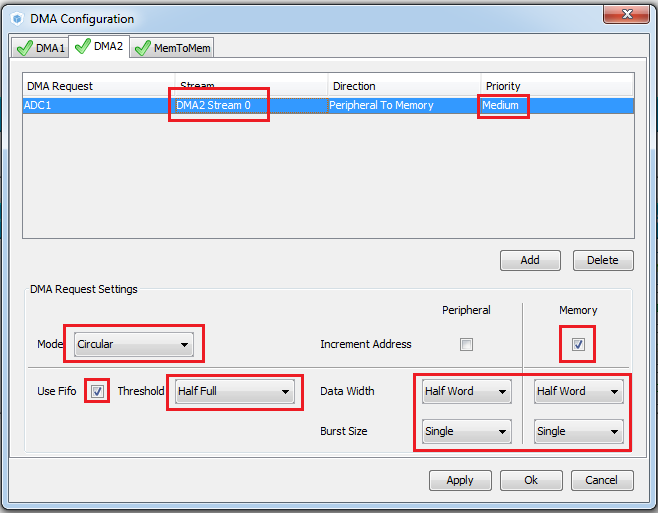
**Lesson 18**

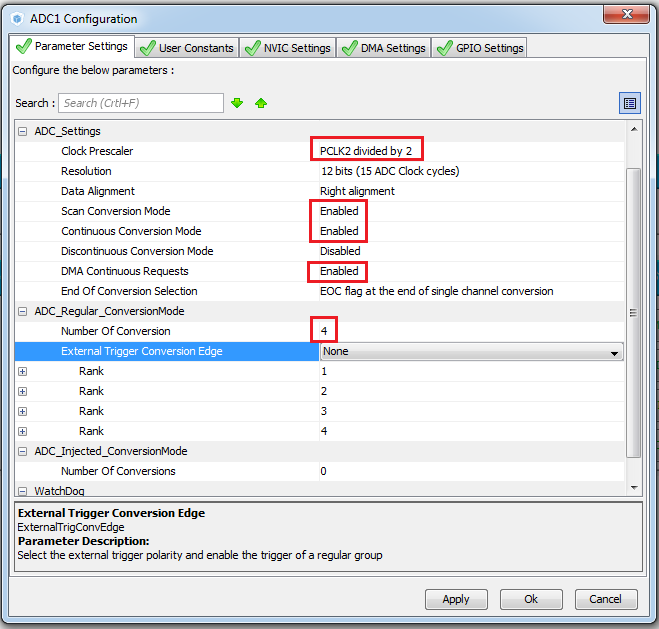
**HAL. ADC. Regular Channel. DMA**

We create the project from **ADC\_REGULAR\_INT** , we call it **ADC\_REGULAR\_DMA** .

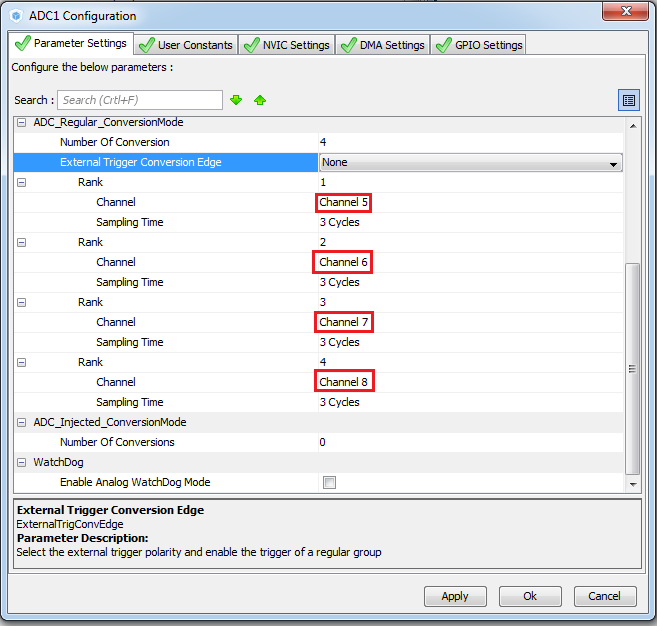
Run Cube, add to our **ADC1** 3 more channels - **INT6, INT8, INT8** . Go to **Configuration** . Interrupts from**ADC** can be disabled for now. Go button **DMA** , there bookmark **DMA2** and include therein **DMA** follows



Apply the changes. We go on the button ADC there we adjust the parameters as follows



Still need to configure the queue channels



We generate and launch our project. We also add lcd.c. Let's assemble it.

Let's fix our variables in data arrays

/ \* Private variables ------------------- \* /

**uint16\_t ADC\_Data [4];**

/ \* USER CODE END PV \* /

 / \* USER CODE BEGIN 1 \* /

**float u [4];**

        char str [9];

  / \* USER CODE END 1 \* /

There, on page 108 of the HAL\_User\_Manual datasheet, you can see how the code is written for the ADC using DMA. Let's correct the start function, well, we'll fix a little delay for the opening of the opening words.

**HAL\_Delay (500);**

        LCD\_Clear ();

        LCD\_SetPos (4, 0);

        LCD\_SendChar ('s');

        LCD\_SetPos (8, 1);

        LCD\_SendChar ('t');

        LCD\_SetPos (12, 2);

        LCD\_SendChar ('m');

        LCD\_SetPos (16, 3);

        LCD\_SendChar ('3');

        LCD\_SendChar ('2');

**HAL\_Delay (500);**

        LCD\_Clear ();

**HAL\_ADC\_Start\_DMA (& hadc1, (uint32\_t \*) & ADC\_Data, 4);**

  / \* USER CODE END 2 \* /

Also, we will use the use of variables in an infinite loop

                u **[0]** = ((float) ADC\_Data **[0]** ) \* 3/4096; // enter the result of the transformations into a variable

                sprintf (str, "%. 2fv", u **[0]** ); // convert the result to a string

                LCD\_SetPos (0,3); // show the result on the LCD display

We remove everything from the function at the bottom of the module.

void HAL\_ADC\_ConvCpltCallback (ADC\_HandleTypeDef \* hadc1)

{

}

We will collect, we will sew, we will check.

Now we need to show data from all voltage sources

To do this, we will make our variable for data not optimizable

**volatile**uint16\_t ADC\_Data [4];

Also, we add a variable to the main function for enumeration of array elements

        char str [9];

**uint8\_t i;**

  / \* USER CODE END 1 \* /

and make some changes in the infinite loop

while (1)

  {

**for (i = 0; i <4; i ++)**

**{**

                        u **[i]** = ((float) ADC\_Data **[i]** ) \* 3/4096; // enter the result of the transformations into a variable

                        sprintf (str, "%. 2fv", u **[i]** ); // convert the result to a string

                        LCD\_SetPos (0, **i** ); // show the result on the LCD display

                        LCD\_String (str);

**}**

Now let's try to count the number of transformations

To do this, add another unoptimized variable to the main module.

volatile uint16\_t ADC\_Data [4];

**volatile uint32\_t cnt1;**

/ \* USER CODE END PV \* /

Initialize it in the main fuknii

  MX\_ADC1\_Init ();

  / \* USER CODE BEGIN 2 \* /

**cnt1 = 0;**

      LCD\_ini ();

Add the increase of this variable to the function of our intercepting the endings of the transformations

/ \* USER CODE BEGIN 4 \* /

void HAL\_ADC\_ConvCpltCallback (ADC\_HandleTypeDef \* hadc1)

{

**cnt1 ++;**

}

/ \* USER CODE END 4 \* /

And we show the counter on the display screen in an infinite loop (the delay can already be removed, it is now physically happening).

                        LCD\_String (str);

                }

**sprintf (str, "% 10u", cnt1); // convert the result to a string**

**LCD\_SetPos (6.0); // show the result on the LCD display**

**LCD\_String (str);**

  / \* USER CODE END WHILE \* /

Let's sew, we'll fill it and see.

Well, for complete clarity, we will need another meter milliseconds from the start of the controller start.

To do this, add another variable

volatile uint32\_t cnt1, **cnt2**;

/ \* USER CODE END PV \* /

We will also initialize it in the main function

        cnt1 = 0; **cnt2 = 0;**

Also connect it to the module stm32f4xx\_it.c

#include "stm32f4xx\_it.h"

/ \* USER CODE BEGIN 0 \* /

**extern volatile uint32\_t cnt2;**

/ \* USER CODE END 0 \* /

/ \* External variables ------------------- \* /

We will increment it in the same module in the function of the system timer

void SysTick\_Handler (void)

{

  / \* USER CODE BEGIN SysTick\_IRQn 0 \* /

**cnt2 ++;**

  / \* USER CODE END SysTick\_IRQn 0 \* /

And also display on the display

                        sprintf (str, "% 10u", cnt1); // convert the result to a string

                        LCD\_SetPos (6.0); // show the result on the LCD display

                        LCD\_String (str);

**sprintf (str, "% 10u", cnt2);**

**LCD\_SetPos (6.1); // show the result on the LCD display**

**LCD\_String (str);**

  / \* USER CODE END WHILE \* /

Let's sew, we'll fill it and see.

In general, if we calculate the frequency by dividing these two indicators, then we will get somewhere around 650 kc, which is quite impressive.

I tried to register the division operation in an infinite loop and display the result on the screen, but, most likely because of a lack of resources, I got a hangman, so we will share in the mind.