**Lesson 20**

**HAL. ADC. Injected Channel. Interrupt**

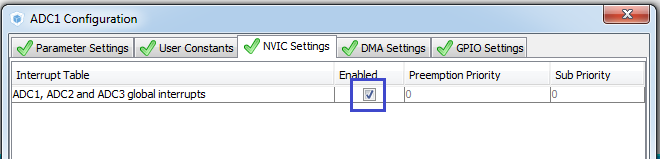
Today we will continue to work with the **ADC's** injection channel .

Only today we will try to work with the interruptions of such a channel, thus we automate the process a little, we will not use delays, which will give our project more professionalism.

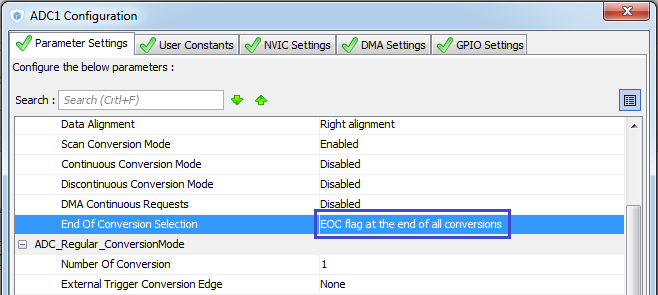
We will also create a project from the previous one - from **ADC\_INJECTED** , and call it **ADC\_INJECTED\_INT** , just typing in the suffix \_INT.

Run the project in the **Cube MX** .

First of all, we, of course, include these very interruptions



Let's go to this tab in the " **Parameter Settings** " tab and switch " **End Of Confersion Selectiom** " to the next state



Apply the settings, generate the project, run it in Keil, configure the programmer for auto-cutting, add the LCD.c file to the project, compile the project and start writing the code.

Copy from an infinite loop a string

 HAL\_ADCEx\_InjectedStart (& hadc1);

and insert it to an endless loop, slightly correcting it by adding the prefix **\_IT**

 LCD\_Clear ();   
**HAL\_ADCEx\_InjectedStart\_IT (& hadc1);**  
  / \* USER CODE END 2 \* /

In an infinite loop, we comment on everything, except for the delay.

We also add an interrupt handler for the injected ADC channel. It almost does not differ from the interrupt handler of the regular

**void HAL\_ADCEx\_InjectedConvCpltCallback (ADC\_HandleTypeDef \* hadc1)   
{**

**}**

The handler from the regular channel will be left for the future.

In the function of the interrupt handler, we will take the result, since it's time to do it here. This handler is called at the end of the stress transformations.

Only convert to a variable type with a floating point here will not be to save processing time. We will use an integer array. We already had it, but in the last lesson we deleted it. Now you have to add it again

/ \* Private variables ------------------- \* /   
**volatile uint16\_t ADC\_Data [4];**

Now we can add the code to the interrupt handler using this array

void HAL\_ADCEx\_InjectedConvCpltCallback (ADC\_HandleTypeDef \* hadc1)   
{   
**ADC\_Data [0] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_1);   
  ADC\_Data [1] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_2);   
  ADC\_Data [2] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_3);   
  ADC\_Data [3] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_4);**  
}

And at the end we need to start the conversion again. You can copy the function call from our code from the **main ()** function , only we do not need an ampersand, since we already have an index

  ADC\_Data [3] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_4);   
  cnt1 ++;   
**HAL\_ADCEx\_InjectedStart\_IT (hadc1);**

}

Add the variable i to the **main ()** function

 char str1 [9];   
**uint8\_t i;**  
  / \* USER CODE END 1 \* /

Now add to the infinite loop a finite loop in which we put all the values ​​read in the interrupt handler into place - in an array of variables with a floating point

while (1)   
{   
**for (i = 0; i <4; i ++) u [i] = ((float) ADC\_Data [i]) \* 3/4096;**

Uncomment the code that we have commented on in an infinite loop and correct it under the new conditions that have been created, displaying the gradually read and processed data on the symbol display

  for (i = 0; i <4; i ++) u [i] = ((float) ADC\_Data [i]) \* 3/4096;   
**sprintf (str, "% .2fv", u [0]); // convert the result to a string   
  sprintf (str1, "%. 2fv", u [1]); // convert the result to   
  strcat (str, str1) ;   
  sprintf (str1, "%. 2fv", u [2]); // convert the result to a string   
  strcat (str, str1);   
  sprintf (str1, "%. 2fv", u [3]); // convert the result to a string   
  strcat (str, str1);   
  LCD\_SetPos (0,0); // show the result on the LCD display   
  LCD\_String (str);**  
  HAL\_Delay (200);   
  / \* USER CODE END WHILE \* /

We will collect our code and check it in practice



We see that everything works fine. You can verify this by turning the resistors in the voltage dividers.

We can also add counters, as we did in the previous year's lesson, and calculate how often our ADC bus works. Even, in principle, you can find that project and copy the code from there, so it will be faster

volatile uint16\_t ADC\_Data [4];   
**volatile uint32\_t cnt1, cnt2;**  
/ \* USER CODE END PV \* /

Also add stm32f4xx\_it.c to the file

/ \* USER CODE BEGIN 0 \* /   
**extern volatile uint32\_t cnt2;**  
/ \* USER CODE END 0 \* /   
......................

void SysTick\_Handler (void)   
{   
  / \* USER CODE BEGIN SysTick\_IRQn 0 \* /   
**cnt2 ++;**  
  / \* USER CODE END SysTick\_IRQn 0 \* /

In any case, initialize the counters in  **main ()**

  / \* USER CODE BEGIN 2 \* /   
**cnt1 = 0; cnt2 = 0;**

We will count in the interrupt handler from the ADC

  ADC\_Data [3] = HAL\_ADCEx\_InjectedGetValue (hadc1, ADC\_INJECTED\_RANK\_4);   
**cnt1 ++;**  
  HAL\_ADCEx\_InjectedStart\_IT (hadc1);   
}

And it remains for us to display this somewhere on the display. But since we took the 1st line, we will display all this in another place

  LCD\_String (str);   
**sprintf (str, "% 10u", cnt1); // convert the result to the line   
  LCD\_SetPos (6,1); // show the result on the LCD display   
  LCD\_String (str);   
  sprintf (str, "% 10u", cnt2); // convert the result to the line   
  LCD\_SetPos (6,2); // show the result on the LCD display   
  LCD\_String (str);**  
  HAL\_Delay (200);   
  / \* USER CODE END WHILE \* /

We will collect the project, we will sew the controller and see the result. We wait, when on the display in the bottom line will be applied will be 10000



And making a simple calculation, we see that the ADC in our work somewhere at a speed of 220 kHz.

It is very good that simultaneously 4 channels manage to convert the data. Inputs at this frequency.

We will also try to adjust the resistors to make sure that we do not hang anything.