CPE3500: Embedded Digital Signal Processing

Lab Exercise 4: Analog to Digital Conversion using ADC and DMA

Objective:

The purpose of this lab exercise is to introduce ADC and Direct Memory Access module configurations in STM32 microcontrollers. The students will be acquiring analog signals generated from a benchtop signal generator.

Introduction:

Analog to digital conversion is the first step in processing a continuous-time signal in a digital system. The theoretical background of analog to digital conversion is discussed in the class. Here, we will start with the basic setup to adjust the ADC module of the STM32L476RG microcontroller. Then, we will analyze the effect of aliasing on a sampled signal.

In this exercise students will:

- Learn how to setup ADC and DMA in STM32CubeIde.
- Create a simple application to record analog data using HAL functions.
- Learn how to generate analog waveforms from a benchtop signal generator.

Required Equipment:

Personal computer (Windows/Mac OS)

Nucleo-L476RG Board (64-pin)

USB Type-A (or Type-C) to Mini-B cable and Jumper Wires

Agilent 33220A Signal Generator

BNC to Alligator Clips Cable

Procedure:

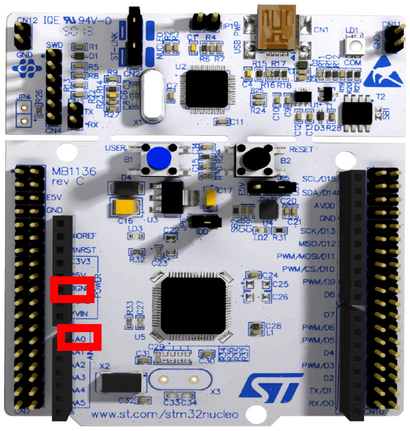
1. Clone a Project in STM32CubeIDE

Clone your project from the previous lab and create a new project following the same steps.

2. Acquire data from STM32 ADC

STM32L476 microcontroller on our Nucelo-L476 board consists of three 12-bit successive approximation (SAR) ADC which each ADC has up to 20 multiplexed channels.

We will use PA0 port pin on the Nucleo board for the analog input.



To generate analog signals, we will use Agilent 33220A signal generators and connect it to PA0 and GND through BNC to alligator clips cable. The BNC connector should be inserted to output port of the signal generator and the red clip should be connected to PA0 through a jumper wire while black clip to GND.

For signal generator settings, we will set Sine wave option with 1 Hz frequency, 3Vpp (3V peak to peak) amplitude and 1.5V offset.

Do not press Output button before making the configurations for the STM32 module!

Open .ioc configuration file in the STM32Cubeide project. Find and click on PA0 port pin in STM32L476 pinout view. Configure PA0 pin as analog input by selecting ADC1_IN5 option.

Then, click on ADC tab on the left menu and ADC1 to open mode and configurations window for ADC1. Select "Single-Ended" option for IN5 channel.

Save and generate the code for this ADC only configuration and switch to C perspective to edit the code.

Inside /* USER CODE BEGIN PV */ section, define two variables as follows:

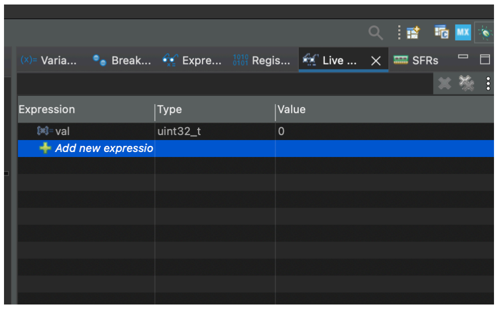
```
uint32_t val;

float real_val;
```

In the /* USER CODE BEGIN 3 */ (inside the while loop), insert the following lines to start and poll ADC conversion and read the value into val variable.

```
HAL_ADC_Start(&hadc1);

HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

val = HAL_ADC_GetValue(&hadc1);

HAL_Delay(100);
```

This code will make ADC conversion every 100 ms and update val variable.

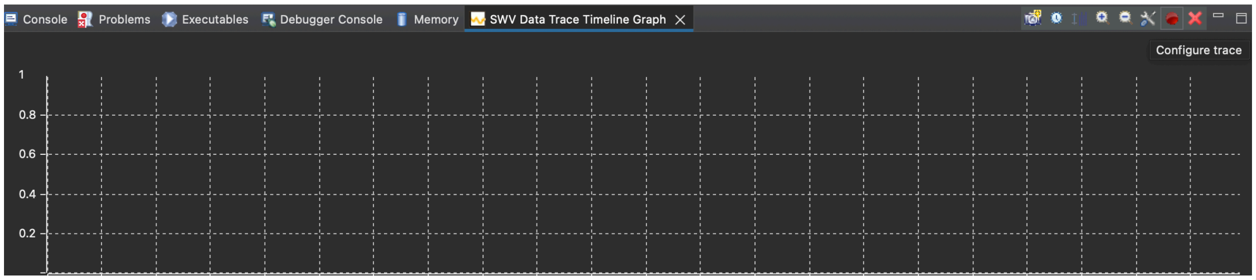
Press Output button on the signal generator and start debugging the code test while observing changes of val variable in Live Expressions window.



Expression	Type	Value
val	uint32_t	0
Add new expressio		

For a graphical monitoring of variables, we will also use Serial Wire Viewer (SWV) window. To activate this, open debug configuration menu and enable SWV.

Then open SWV DataTrace tab at the bottom window and click Configure Trace button to make data trace configurations for val variable.



Debug the code and monitor live plot for the val variable.

<u>Task-1:</u>

Change the delay to 50 ms. Define another variable as float to calculate the voltage equivalent of the acquired ADC data by making proper conversion. Take screenshot of the SWV window.

Define a float type recording array of size 40. In the while loop, save the first 40 ADC readings into this array (by using a counter). Place a breakpoint after the buffer is full and export the data.

Repeat the same experiment for 5 Hz and 15Hz. Plot all three waveforms in Matlab. Compare and comment on the results.

3. Measuring A/D Conversion Time with Polling

Open Configuration file. Click on Timers tab and select TIM6. From the mode and configuration menu, activate the timer and set the prescaler to 79 to generate 1 us timer interval.
In the code, first define the following variable:

```
uint32_t timer_counter;
```

Inside /* USER CODE BEGIN 2 */, insert the line below to start Timer 6:

```
HAL_TIM_Base_Start(&htim6);
```

Inside the while loop insert this line before HAL_ADC_Start line inside while loop:
htim6.Instance->CNT=0;

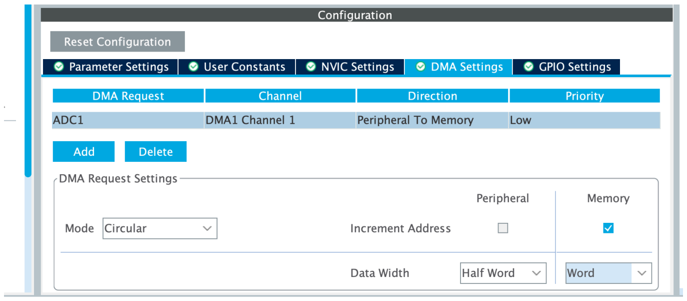
Then, insert the below line after getting the ADC value:
timer_counter = __HAL_TIM_GET_COUNTER(&htim6);

Task-2:

Run the code in debug mode and monitor the timer_counter variable in Live Expressions window. Calculate the maximum sampling rate of the ADC according to this measurement.

4. A/D Conversion with Direct Memory Access (DMA)

Open Configuration file. Click on Analog tab and select ADC1. In the configuration menu, click on DMA settings and add DMA. Select ADC1 and make Mode as Circular. Change the Data Width into Word.

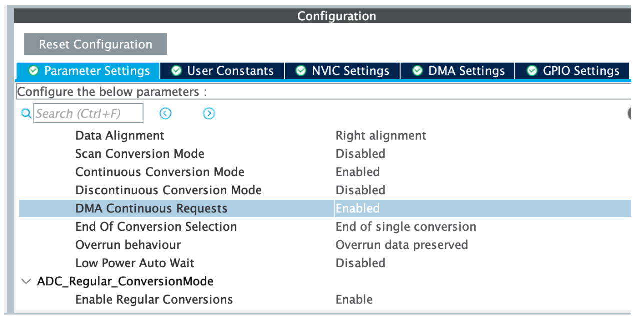


DMA Request	Channel	Direction	Priority
ADC1	DMA1 Channel 1	Peripheral To Memory	Low

Mode: Circular

	Peripheral	Memory
Increment Address	☐	✓
Data Width	Half Word	Word

After completing DMA settings, click on Parameter Settings of the ADC and make Continuous Conversion mode and DMA Continuous Requests configurations enabled.



Data Alignment	Right alignment
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Enabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Enabled
End Of Conversion Selection	End of single conversion
Overrun behaviour	Overrun data preserved
Low Power Auto Wait	Disabled
ADC_Regular_ConversionMode	
Enable Regular Conversions	Enable

Save and switch to C perspective.

Define a constant as below:

#define buffer_size 80

Define adc buffer for DMA:

uint32_t adc_buffer[buffer_size];

Inside /* USER CODE BEGIN 2 */, insert the line below to start DMA recording:

```
HAL_ADC_Start_DMA(&hadc1, adc_buffer, buffer_size);
```

Comment all the lines inside the while loop from previous tasks.

Inside /* USER CODE BEGIN 4 */, define two callback functions as below:

```
void HAL_ADC_ConvHalfCpltCallback(ADC_HandleTypeDef *hadc)
{
        htim6.Instance->CNT=0;
        val = adc_buffer[0];
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc)
{
        timer_counter = __HAL_TIM_GET_COUNTER(&htim6);
}
```

Task-3:

Run the code in debug mode while monitoring timer_counter value. Calculate the A/D conversion sampling rate in DMA case.

Place a breakpoint val = adc_buffer[0] line above to record ADC readings through DMA (for half of the buffer with a length of 40) and debug the code for 1 Hz sine frequency. Export the data and plot it in Matlab.

Repeat the same experiment for 100 kHz. Export and plot the data in Matlab. Compare and comment on the results.

Lab Exercise 4 Report:

Prepare a lab report (as single pdf file) consisting of the followings and upload it to the D2L dropbox.

- Cover Sheet
- Task-1: Screenshot of the SWV window. All the plots with proper labeling and title in Matlab. Comments

- Task-2: Live Expression screenshot and sampling rate calculation.
- Task-3: Modified sampling rate calculation in DMA mode and Matlab plots with comparisons. Comment on the results.
- Conclusion (1-paragraph)