

ADC with STM32CubeMX and HAL

This tutorial will demonstrate how to utilize STM32CubeMX tool to initialize peripherals, build and generate C code using HAL libraries.

After this tutorial you will be able to:

- Create and configure STM32CubeMX project and generate initialization code
- Program and use HAL functions to collect ADC data from a selected analog channel (i.e. internal temperature sensor)

Hardware:

- Nucleo-L476RG board(64-pin),available at: www.st.com/en/evaluation-tools/nucleo-l476rg.html
- Standard-A -to- Mini USB cable

Literature:

- [STM32L476xx Datasheet](#)
- [UM1724](#) User manual STM32 Nucleo-64 boards
- [UM1884](#) Description of STM32L4/L4+ HAL and low-layer drivers
- [UM1718](#) User manual STM32CubeMX for STM32 configuration and initialization C code generation

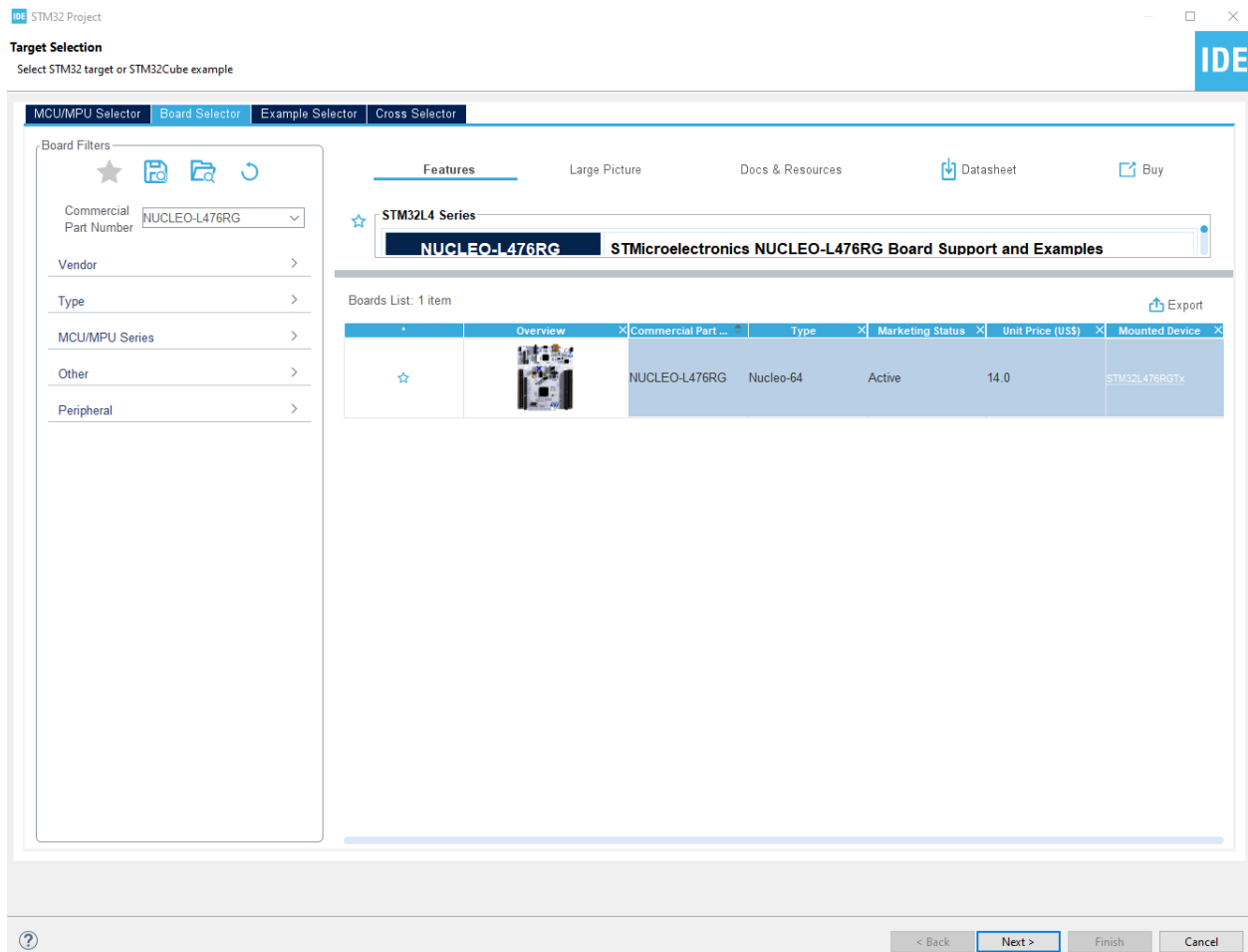
Stages

- 1: Create New Project with STM32CubeMX
- 2: Pinout Configuration
- 3: Clock Configuration
- 4: Configure project and Generate Source Code
- 5: Edit main.c
- 6: Build Project
- 7: Debug the Project



1: CREATE NEW PROJECT USING STM32CUBEMX:

- Open STM32CubeIDE
- Click *File -> New -> STM32 Project*. A target selection window will open.
- From Board Selector type *Nucleo-L476RG*. Select the board and click next.
- Name your project “Nucleo_L476RG_ADC” and click Finish.
- Answer “Yes” to “Initialize all peripherals with their default mode?” popup.



2: Pinout Configuration

Under Analog select ADC1. We want to enable Temperature Sensor Channel to read from the internal temperature sensor. Next enable *Continuous Conversion Mode*

The screenshot displays the STM32CubeMX configuration tool. On the left, the 'Analog' tab is selected, and 'ADC1' is highlighted. The main panel shows the configuration for ADC1. The 'Temperature Sensor Channel' checkbox is checked and highlighted with a red box. Below this, the 'EXTI Conversion Trigger' is set to 'Disable'. A 'Configuration' panel is open, showing a tree view of settings. The 'Parameter Settings' tab is selected. In the 'ADC_Settings' section, the 'Continuous Conversion Mode' is set to 'Enabled' and is highlighted with a red box. Other settings like 'Mode' (Independent mode), 'Clock Prescaler' (Asynchronous clock mode divided ...), 'Resolution' (ADC 12-bit resolution), 'Data Alignment' (Right alignment), 'Scan Conversion Mode' (Disabled), 'Discontinuous Conversion' (Disabled), 'DMA Continuous Requests' (Disabled), 'End Of Conversion Selection' (End of single conversion), 'Overrun behaviour' (Overrun data preserved), and 'Low Power Auto Wait' (Disabled) are visible. The 'ADC_Regular_ConversionMode' is also set to 'Enabled'.

System Core >

Analog >

- ADC1
- ADC2
- ADC3
- COMP1
- COMP2
- DAC1
- OPAMP1
- OPAMP2

Timers >

Connectivity >

Multimedia >

Security >

Computing >

Middleware >

IN11 Disable

IN12 Disable

IN13 Disable

IN14 Disable

IN15 Disable

☐ IN16 Single-ended

☒ Temperature Sensor Channel

☐ Vbat Channel

☐ Vrefint Channel

EXTI Conversion Trigger Disable

Configuration

Reset Configuration

NVIC Settings DMA Settings

Parameter Settings User Constants

Search (Ctrl+F)

ADCs_Common_Settings

Mode Independent mode

ADC_Settings

Clock Prescaler Asynchronous clock mode divided ...

Resolution ADC 12-bit resolution

Data Alignment Right alignment

Scan Conversion Mode Disabled

Continuous Conversion ... Enabled

Discontinuous Conversion ... Disabled

DMA Continuous Requests Disabled

End Of Conversion Selec... End of single conversion

Overrun behaviour Overrun data preserved

Low Power Auto Wait Disabled

ADC_Regular_ConversionMode

Enable Regular Conversion

Under Connectivity, select USART2 and verify it is configured match the below image.

USART2 Mode and Configuration

Mode

Mode Asynchronous ▼

Hardware Flow Control (RS232) Disable ▼

☐ Hardware Flow Control (RS485)

Configuration

Reset Configuration

✓ NVIC Settings

✓ DMA Settings

✓ GPIO Settings

✓ Parameter Settings

✓ User Constants

Configure the below parameters :

🔍

⏪ ⏩ ℹ

▼ Basic Parameters

Baud Rate	115200 Bits/s
Word Length	8 Bits (including Parity)
Parity	None
Stop Bits	1

▼ Advanced Parameters

Data Direction	Receive and Transmit
Over Sampling	16 Samples
Single Sample	Disable

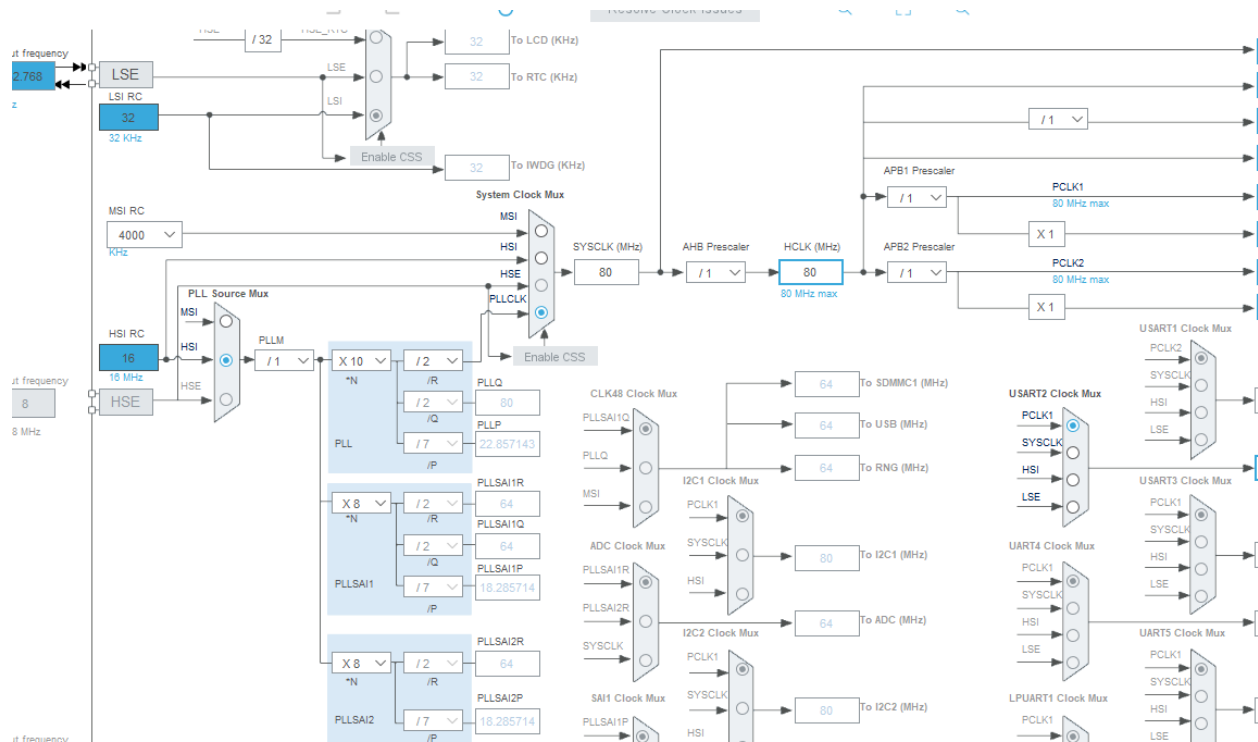
▼ Advanced Features

Auto Baudrate	Disable
TX Pin Active Level I...	Disable
RX Pin Active Level ...	Disable
Data Inversion	Disable
TX and RX Pins Sw...	Disable
Overrun	Enable
DMA on RX Error	Enable
MSB First	Disable

3. CLOCK CONFIGURATION

In the clock configuration tab you can see that STM32CubeMX automatically configures the internal oscillator in the clock system with PLL @80MHz. The HSI is selected as the PLL source and the PLLCLK is selected in the system clock mux.

HCLK is set to 80 MHz.



4: GENERATE CODE

We can now generate code. Click File->Save. You will be asked to generate code, press yes.

Under the project explorer navigate to *Core->Src->main.c*.

5: EDIT main.c

At the top of the file, add:

#include <string.h> and *#include<stdio.h>*

Referring to the UM1884 User manual we can use *HAL_ADC_PollForConversion* to wait for conversion to be completed. We can specify a timeout value in milliseconds. We can use *HAL_MAX_DELAY* to wait for a very long time.

HAL_ADC_PollForConversion

Function name

HAL_StatusTypeDef HAL_ADC_PollForConversion (ADC_HandleTypeDef * hadc, uint32_t Timeout)

Function description

Wait for regular group conversion to be completed.

Parameters

- **hadc**: ADC handle
- **Timeout**: Timeout value in millisecond.

Return values

- **HAL**: status

HAL_ADC_GetValue will give us the raw conversion value. We can then use *HAL_UART_Transmit* to print our data to the terminal.

HAL_ADC_GetValue

Function name

uint32_t HAL_ADC_GetValue (ADC_HandleTypeDef * hadc)

Function description

Get ADC regular group conversion result.

Parameters

- **hadc**: ADC handle

Return values

- **ADC**: group regular conversion data

From the STM32L476xx datasheet we can get values for our raw value to temperature conversion.

- 4095 is max value for 12 bit conversion
- 3300 is reference voltage in millivolts
- 760 is voltage in millivolts at 30°C
- 2.5 is average slope in mV/°C

In int main():

```
/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_USART2_UART_Init();
MX_ADC1_Init();
/* USER CODE BEGIN 2 */

/* Initialize all configured peripherals */
MX_ADC1_Init();

HAL_ADCEX_Calibration_Start(&hadc1, ADC_SINGLE_ENDED);

HAL_ADC_Start(&hadc1);

while (1) {
    char msg[20];
    uint16_t rawValue;
    float temp;

    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);

    rawValue = HAL_ADC_GetValue(&hadc1);
    temp = ((float)rawValue) / 4095 * 3300;
    temp = ((temp - 760.0) / 2.5) + 30;


    sprintf(msg, "rawValue: %hu\r\n", rawValue);
    HAL_UART_Transmit(&huart2, (uint8_t*) msg, strlen(msg), HAL_MAX_DELAY);

    sprintf(msg, "Temperature: %f\r\n", temp);
    HAL_UART_Transmit(&huart2, (uint8_t*) msg, strlen(msg), HAL_MAX_DELAY);
}
/* USER CODE END 3 */
}
```

6: BUILD THE PROJECT

Connect your USB cable from the computer to your Nucleo Board. Right click the project from the project explorer and click “Build project” to compile the project.

7: DEBUG THE PROJECT

Click on the Debug toolbar icon to start the debug session. Another way to debug is to *Run->Debug* . 

Click the Resume icon to continue the execution. Open a serial monitor such as Putty and select your com port with the appropriate baud rate (115200). We can now see the raw and converted temperature values.

