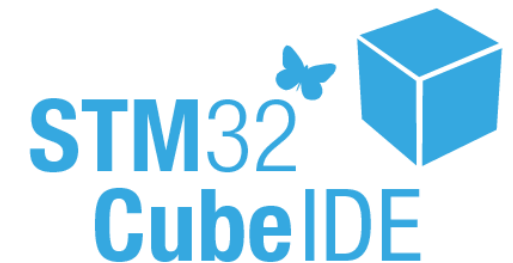


STM32CubeIDE basics

ADC lab: DMA + TIM configuration using HAL library



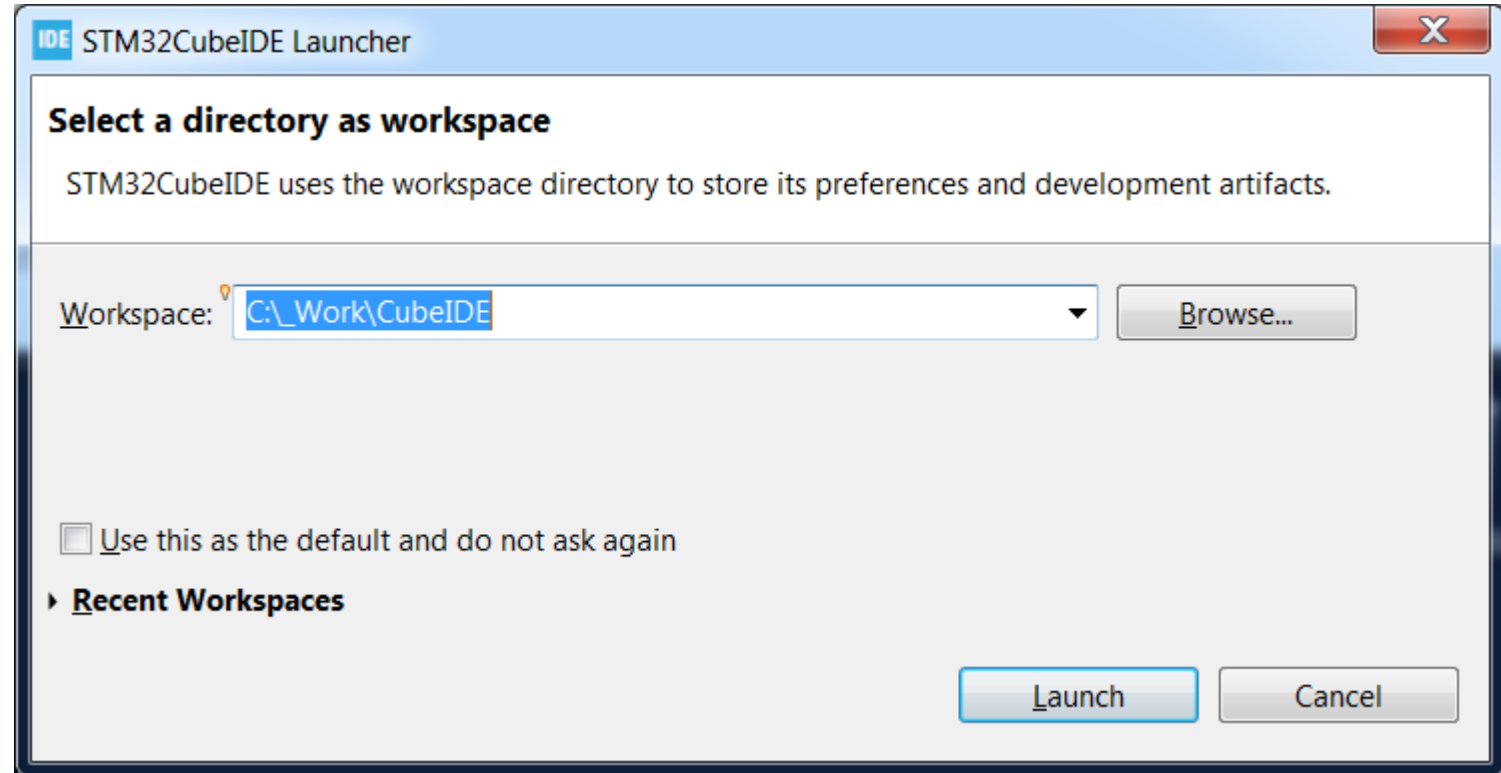
Lab: ADC+DMA+TIM configuration

Objective:

- Using ADC collect data from selected analog channel (i.e. internal temperature sensor),
- Each conversion should be triggered by TIM2 each 1 second
- Data should be transferred via DMA to internal buffer (8 results long)
- When buffer is full, TIM2 should be stopped and interrupt should be raised to allow ADC data postprocessing

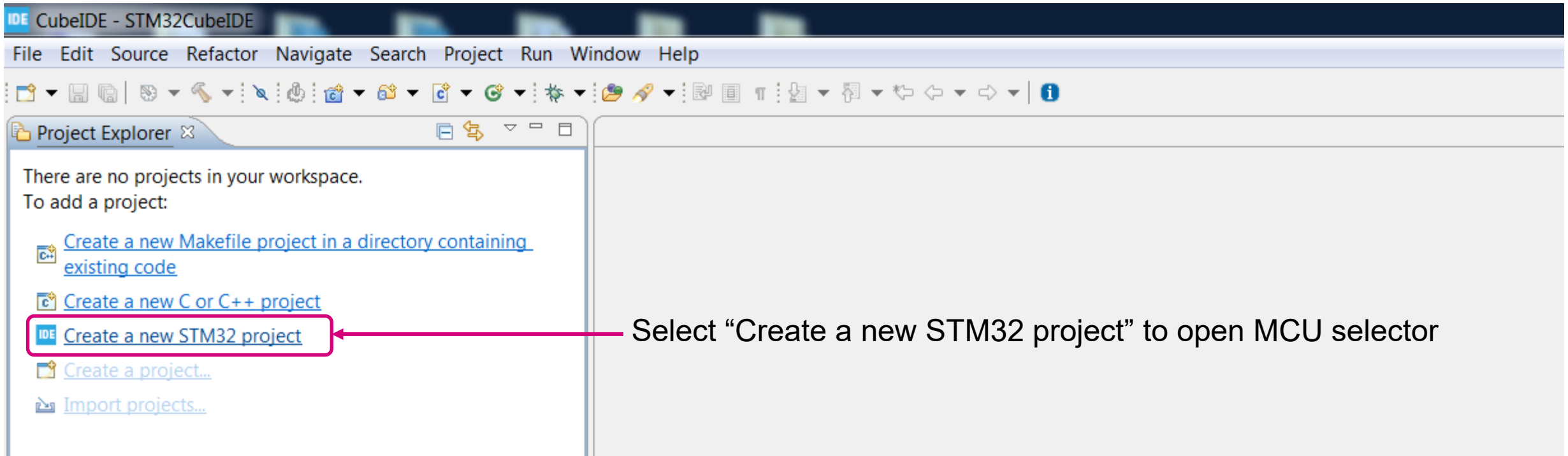
Start a new workspace

- Run STM32CubeIDE
- Select a folder to store a workspace



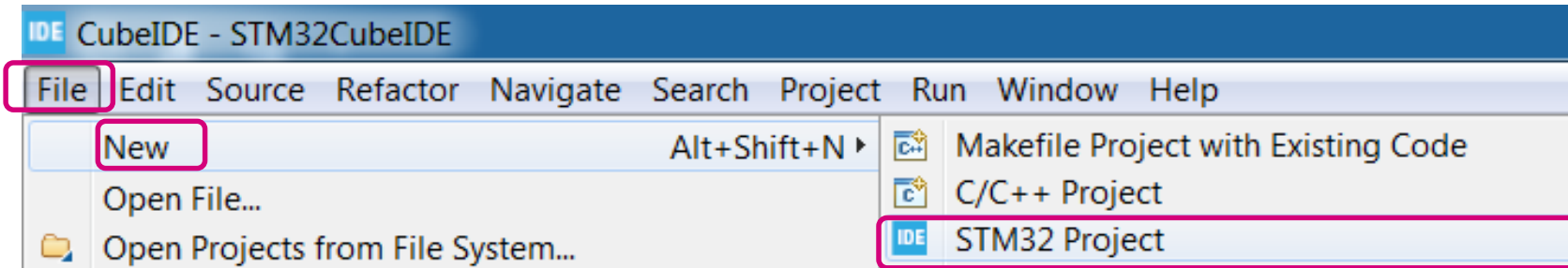
Create a new project

- Click on “Create a new STM32 project”

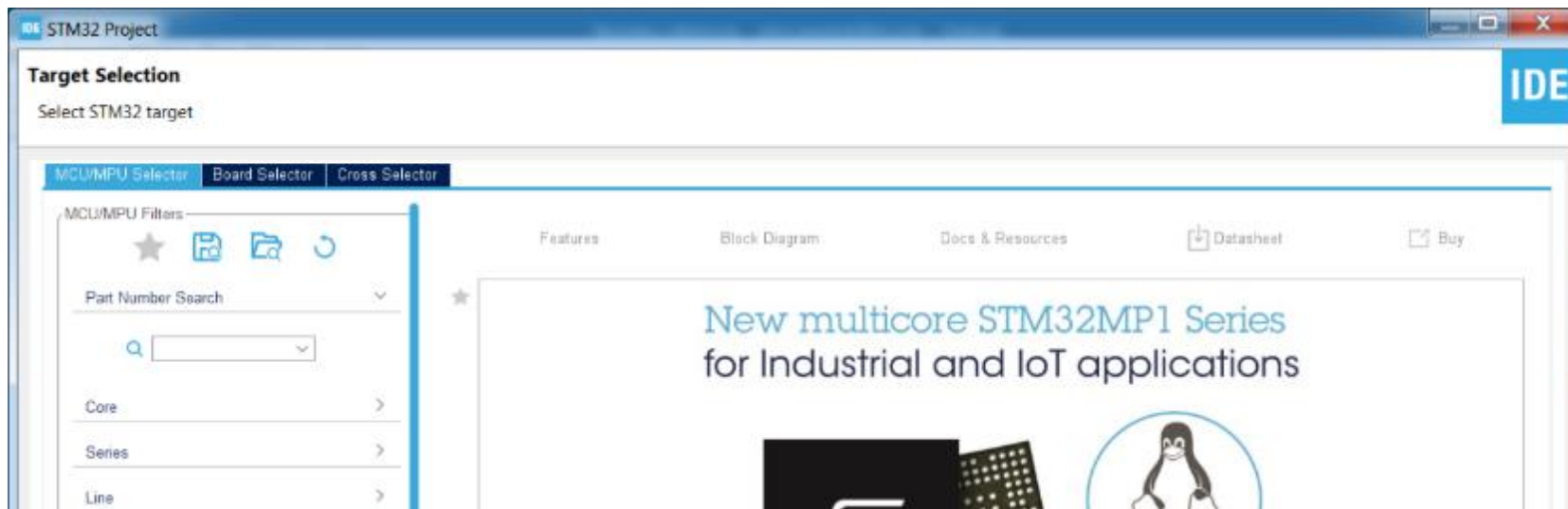


Start a new project within workspace

- Go to File -> New -> STM32 Project

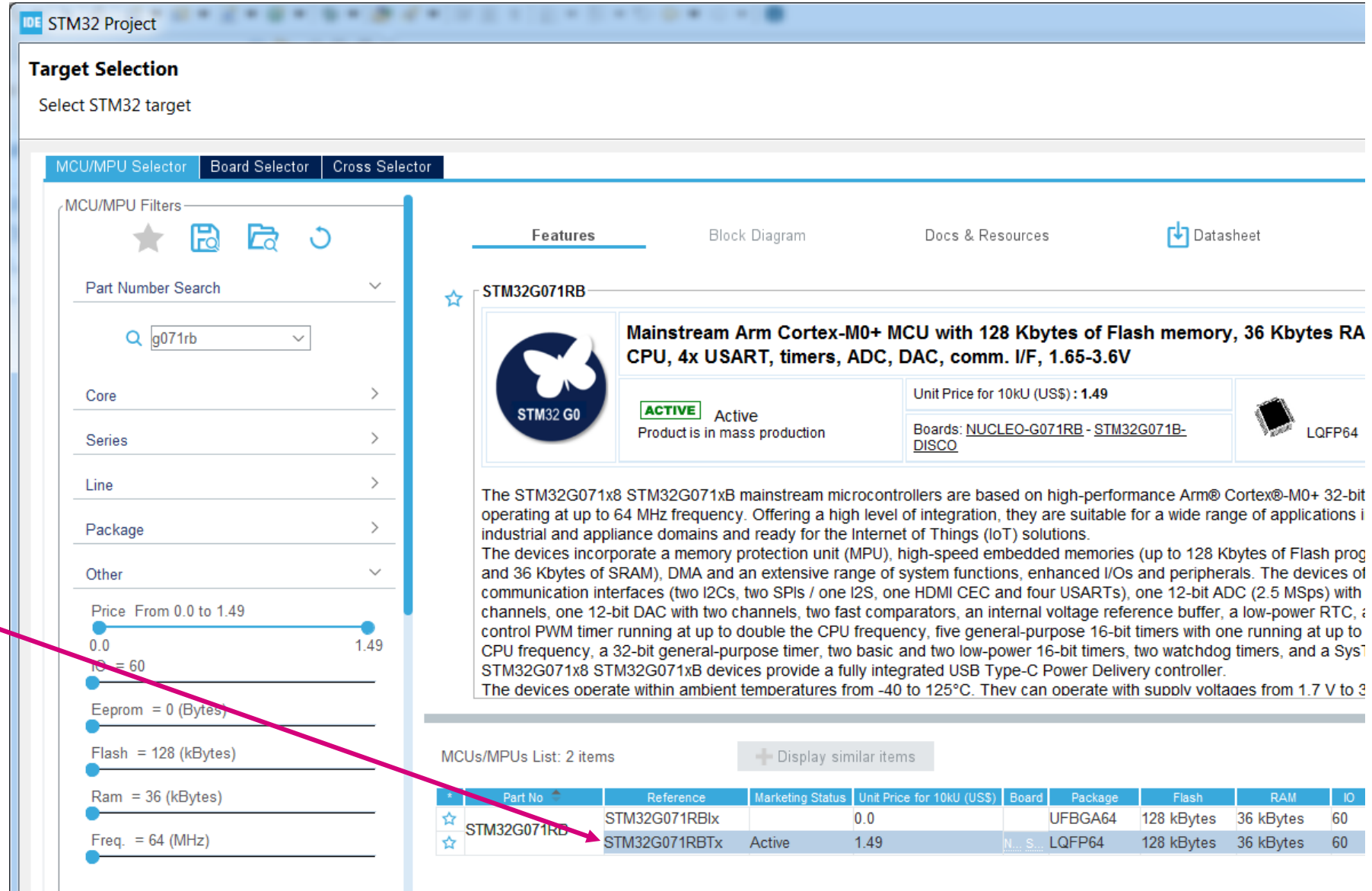


- As a result “target selection” window will be displayed



Select target MCU: STM32G071RBTx

- It is possible to view on main MCU features, download its documentation
- To start a new project we need to double click on the part number



Target Selection
Select STM32 target

MCU/MPU Selector | Board Selector | Cross Selector

MCU/MPU Filters

Part Number Search: g071rb

Core: >

Series: >

Line: >

Package: >

Other: >

Price: From 0.0 to 1.49

IO = 60

Eeprom = 0 (Bytes)

Flash = 128 (kBytes)

Ram = 36 (kBytes)

Freq. = 64 (MHz)

STM32G071RB

Mainstream Arm Cortex-M0+ MCU with 128 Kbytes of Flash memory, 36 Kbytes RAM, CPU, 4x USART, timers, ADC, DAC, comm. I/F, 1.65-3.6V

ACTIVE Active
Product is in mass production

Unit Price for 10kU (US\$): 1.49

Boards: [NUCLEO-G071RB](#) - [STM32G071B-DISCO](#)

LQFP64

The STM32G071x8 STM32G071xB mainstream microcontrollers are based on high-performance Arm® Cortex®-M0+ 32-bit operating at up to 64 MHz frequency. Offering a high level of integration, they are suitable for a wide range of applications in industrial and appliance domains and ready for the Internet of Things (IoT) solutions.

The devices incorporate a memory protection unit (MPU), high-speed embedded memories (up to 128 Kbytes of Flash program and 36 Kbytes of SRAM), DMA and an extensive range of system functions, enhanced I/Os and peripherals. The devices of communication interfaces (two I2Cs, two SPIs / one I2S, one HDMI CEC and four USARTs), one 12-bit ADC (2.5 MSps) with channels, one 12-bit DAC with two channels, two fast comparators, an internal voltage reference buffer, a low-power RTC, a control PWM timer running at up to double the CPU frequency, five general-purpose 16-bit timers with one running at up to CPU frequency, a 32-bit general-purpose timer, two basic and two low-power 16-bit timers, two watchdog timers, and a SysT

STM32G071x8 STM32G071xB devices provide a fully integrated USB Type-C Power Delivery controller.

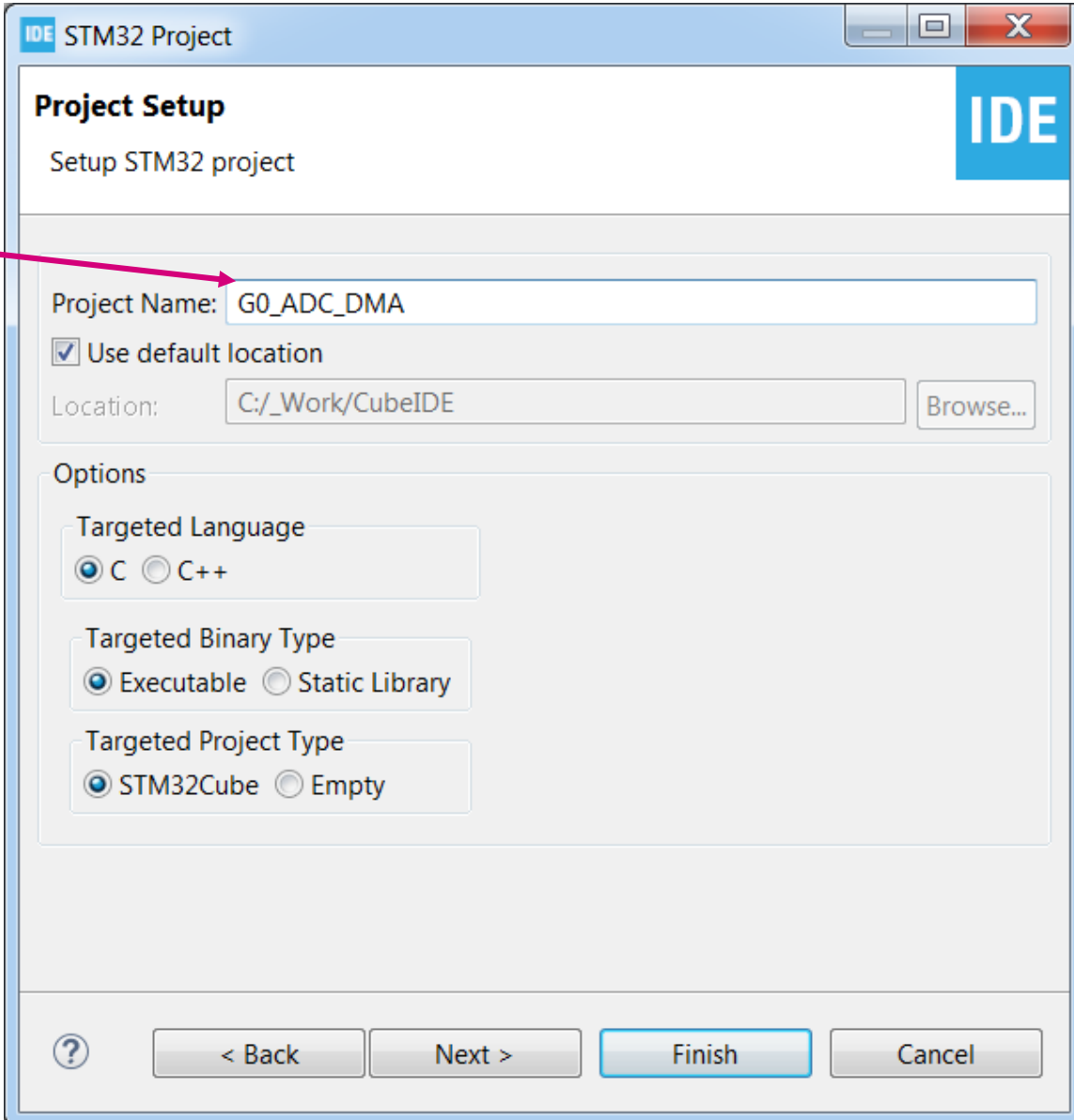
The devices operate within ambient temperatures from -40 to 125°C. They can operate with supply voltages from 1.7 V to 3

MCUs/MPUs List: 2 items

	Part No	Reference	Marketing Status	Unit Price for 10kU (US\$)	Board	Package	Flash	RAM	IO
☆	STM32G071RBx	STM32G071RBx		0.0		UFPGA64	128 kBytes	36 kBytes	60
☆	STM32G071RBTx	STM32G071RBTx	Active	1.49	N... S...	LQFP64	128 kBytes	36 kBytes	60

Enter project name

- Specify project name, optionally its location (if different from workspace one)
- Additionally we can specify target language (C or C++), binary type (executable or static library) and project type (generated by STM32CubeMX or an empty one)



IDE STM32 Project

Project Setup

Setup STM32 project

Project Name: G0_ADC_DMA

☒ Use default location

Location: C:/_Work/CubeIDE Browse...

Options

Targeted Language

☒ C ☐ C++

Targeted Binary Type

☒ Executable ☐ Static Library

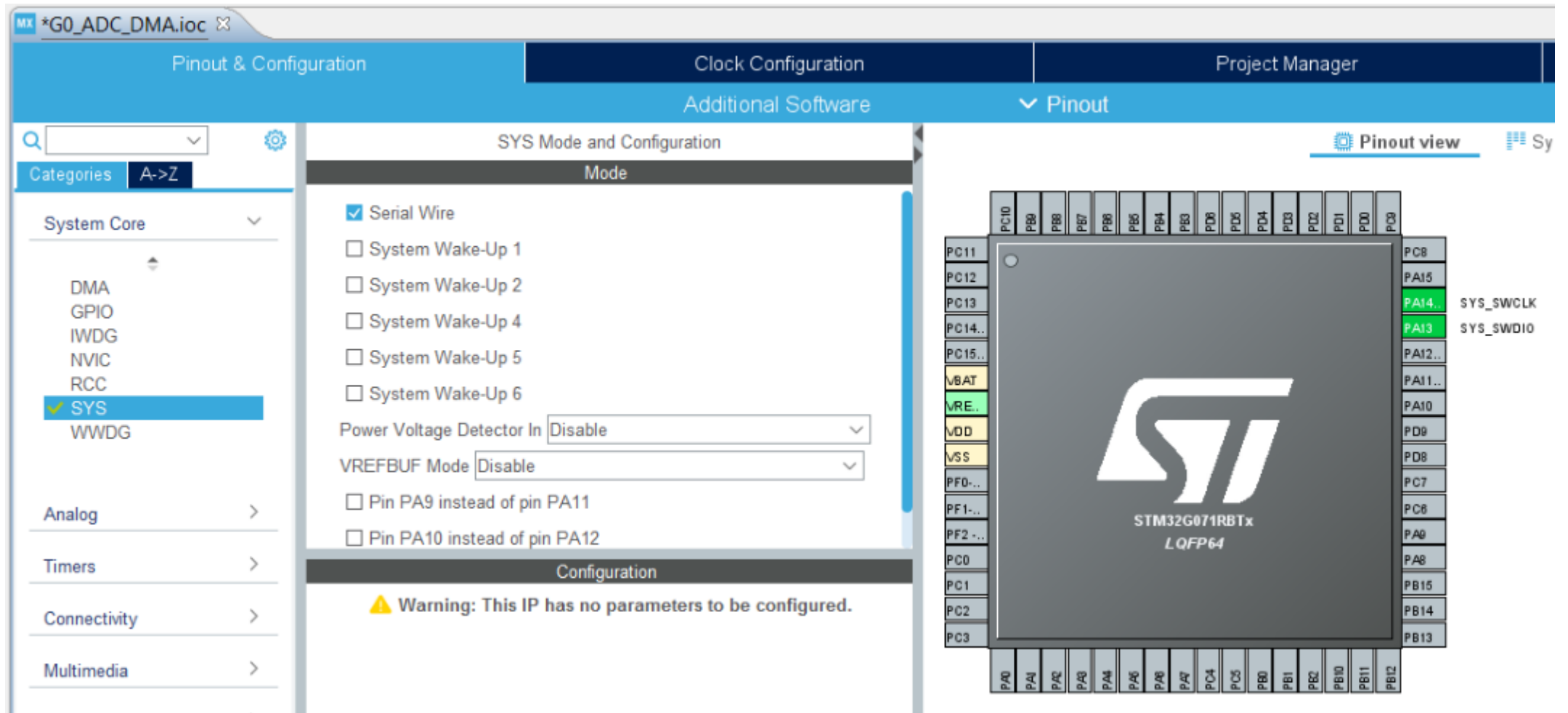
Targeted Project Type

☒ STM32Cube ☐ Empty

? < Back Next > Finish Cancel

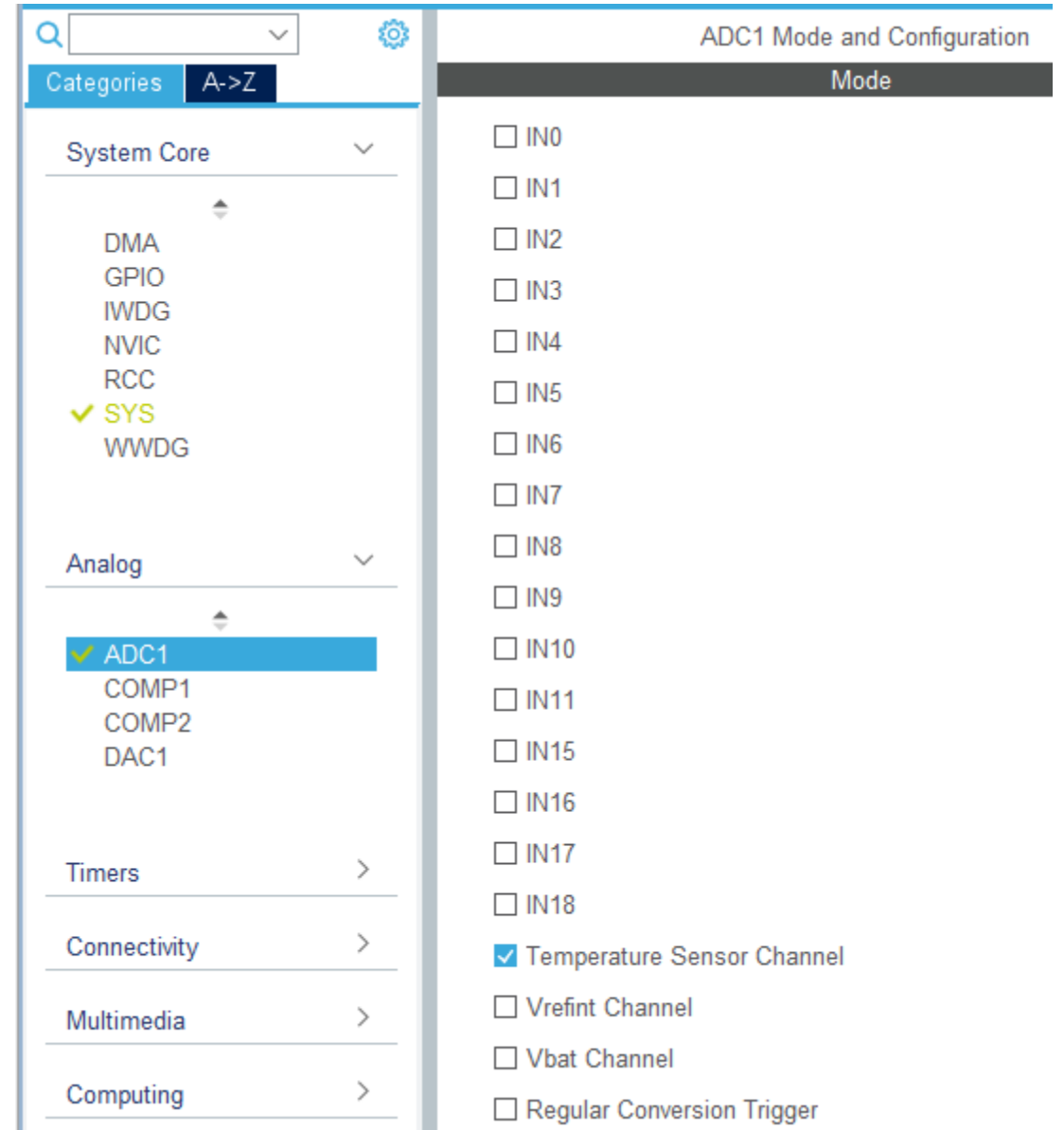
Enabling Serial Wire debug interface

- Select “Serial Wire” from System Core -> SYS peripheral group
- As a result PA13 and PA14 will be assigned to SWD interface



- Select ADC1 within Analog peripherals group and enable “Temperature Sensor Channel” within Mode window

ADC channel selection



- In ADC1 configuration, DMA Settings tab, please Add a new DMA request
- Select ADC1 from the list
- Set its configuration to:
 - Normal Mode
 - Increment on Memory side (data buffer)
 - Half Word data width on both sides (as we will operate on 12bit data)

ADC+DMA configuration



Parameter Settings | User Constants | NVIC Settings | **DMA Settings**

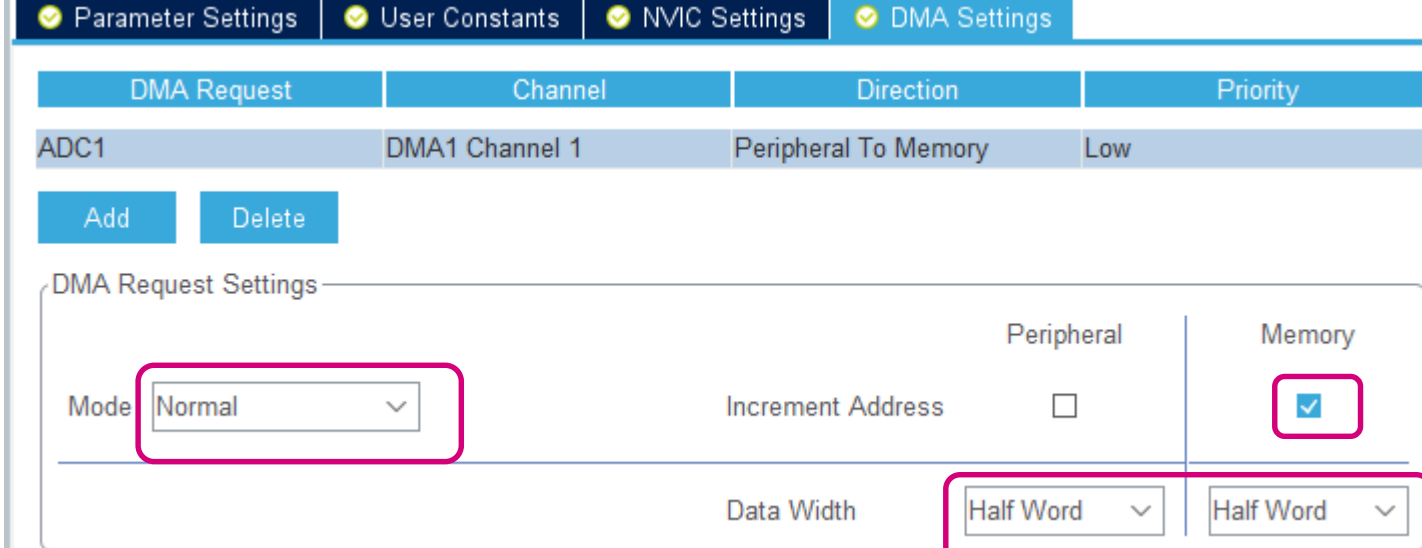
DMA Request	Channel	Direction
Select		

Add Delete



Parameter Settings | User Constants | NVIC Settings | **DMA Settings**

DMA Request	Channel	Direction
Select		
Select		
ADC1		



Parameter Settings | User Constants | NVIC Settings | **DMA Settings**

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

Add Delete

DMA Request Settings

	Peripheral	Memory
Mode	Normal	<input checked="" type="checkbox"/>
Increment Address	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Data Width	Half Word	Half Word

ADC configuration

- In ADC1 configuration, Parameter Settings tab configure:
 - DMA Continuous Requests: Disabled (as we are not using Circular DMA mode)
 - SamplingTime Common1 to 79.5 cycles
 - External Trigger Source: Timer2 Trigger Out event (this event is programmed within Timer2 configuration)
 - External Trigger Edge: rising edge
 - Sampling time common 1 as sampling time for temperature sensor channel

Parameter Settings | User Constants | NVIC Settings | DMA Settings

Configure the below parameters :

Search (Ctrl+F) < >

ADC_Settings	
Clock Prescaler	Synchronous clock mode divided by 2
Resolution	ADC 12-bit resolution
Data Alignment	Right alignment
Sequencer	Sequencer set to fully configurable
Scan Conversion Mode	Disabled
Continuous Conversion Mode	Disabled
Discontinuous Conversion Mode	Disabled
DMA Continuous Requests	Disabled
End Of Conversion Selection	End of single conversion
Overrun behaviour	Overrun data preserved
Low Power Auto Wait	Disabled
Auto Off	Disabled
Oversampling Mode	Disabled
ADC_Regular_ConversionMode	
SamplingTime Common 1	79.5 Cycles
SamplingTime Common 2	1.5 Cycles
Number Of Conversion	1
External Trigger Conversion Source	Timer 2 Trigger Out event
External Trigger Conversion Edge	Trigger detection on the rising edge
Trigger Frequency	High frequency
Rank	
Channel	Channel Temperature sensor
Sampling Time	Sampling time common 1

Timer Parameters Calculation

- TIM2 will be used to trigger ADC1 conversions each 1second
- System is working on default clock settings (HSI 16MHz, no PLL) and we use ADC clock as system clock / 2 \Rightarrow 8MHz
- We have Timer2 input clock 16 MHz (default system clock settings using HSI without PLL)
- We can set prescaler (PSC) to 15999 (actual divide is PSC+1) to have 1kHz on Timer2 counter input
- Then we need to set Timer2 period to 999 (as we are counting from 0) to have overflow (update event) each 1 second

TIM2 configuration

- Select Timers->TIM2
- Set clock source to Internal Clock
- Configure:
 - Prescaler to 15999
 - Counter period to 999
 - MSM bit: Enable
 - TRGO selection: Update Event



TIM2 Mode and Configuration

Mode

Slave Mode

Trigger Source

Clock Source

Configuration

Reset Configuration

Parameter Settings User Constants NVIC Settings DMA Settings

Configure the below parameters :

Search (Ctrl+F)

Counter Settings

Prescaler (PSC - 16 bits value)

Counter Mode

Counter Period (AutoReload Register - 32 bit..)

Internal Clock Division (CKD)

auto-reload preload

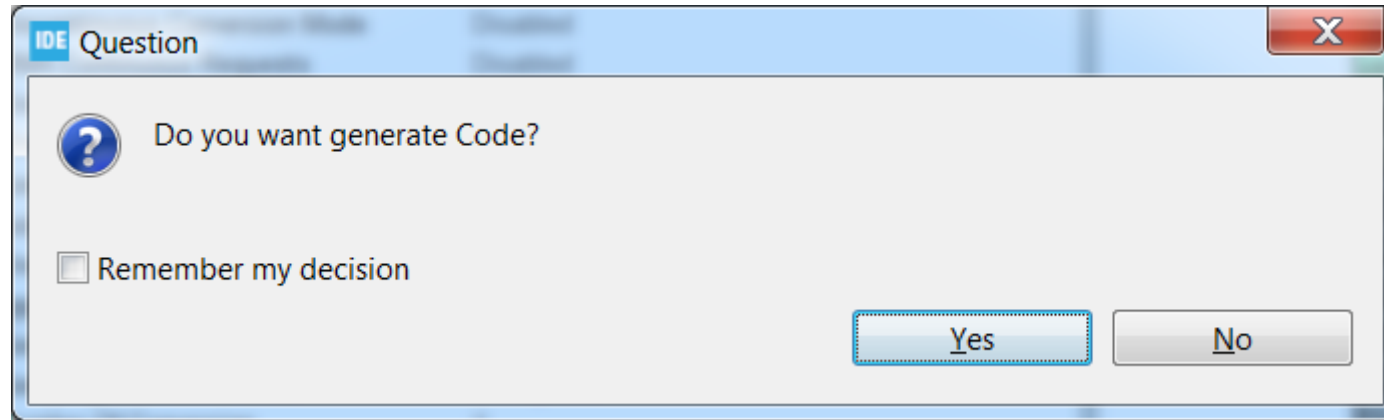
Trigger Output (TRGO) Parameters

Master/Slave Mode (MSM bit)

Trigger Event Selection TRGO

Generate the code

- Save the design settings to trigger code generation



- Define data buffer size and ADC reference voltage (in mV) used for temperature calculation

```
/* USER CODE BEGIN PD */
#define ADC_BUF_SIZE      8
#define __VREFANALOG_VOLTAGE__ 3300//ADC reference voltage in mV
```

- Define data buffer for temperature measurements

```
/* USER CODE BEGIN PV */
uint16_t ADC_buffer[8];
```

- Perform ADC calibration, start ADC and start TIM2 which would trigger ADC conversion each 1second

```
/* USER CODE BEGIN 2 */
if(HAL_ADCEx_Calibration_Start(&hadc1) != HAL_OK)
    Error_Handler();

if(HAL_ADC_Start_DMA(&hadc1 , (uint32_t *)ADC_buffer, ADC_BUF_SIZE) != HAL_OK)
    Error_Handler();

if(HAL_TIM_Base_Start(&htim2) != HAL_OK)
    Error_Handler();
```

- In all cases we are monitoring proper function execution by checking return value

- As we have configured DMA in Normal mode, it would be stopped once the buffer will be filled with the data
- Additionally a DMA transfer complete interrupt will be triggered which could be used to stop ADC or simply TIM2 which is triggering ADC.
- DMA transfer complete interrupt is linked with `HAL_ADC_ConvCpltCallback()`, so we can use the same callback in case we are using ADC in IRQ or in DMA mode

```
/* USER CODE BEGIN 4 */
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if(HAL_ADC_Stop_DMA(&hadc1) != HAL_OK)
        Error_Handler();
}
/* USER CODE END 4 */
```

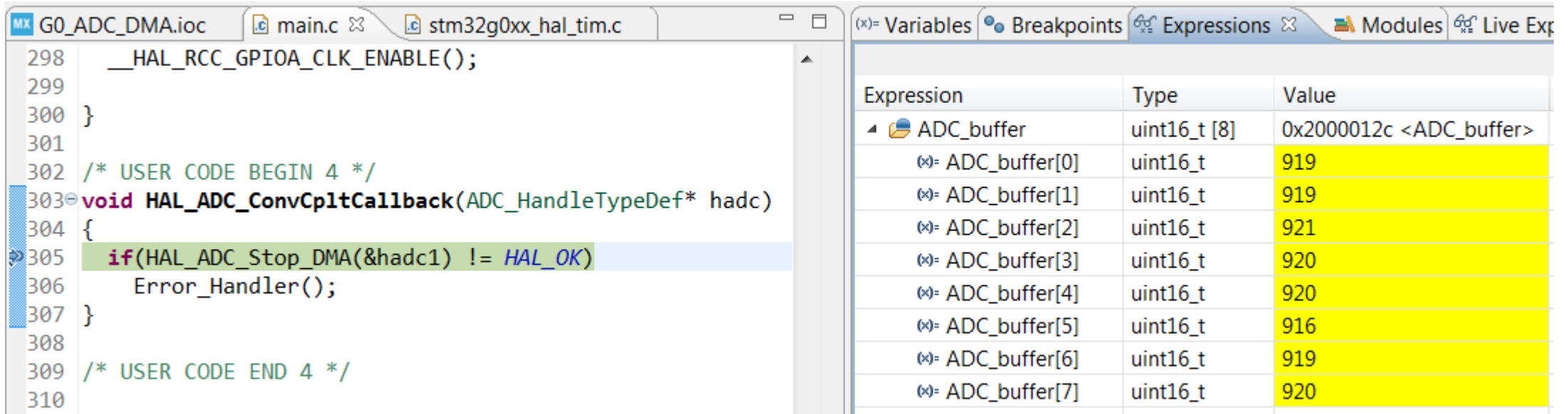
... Let's check it



- After all code processing we can build the project, start debug session and run the application
- We can monitor ADC_buffer (while stopped on breakpoint or paused the debug session) by:
 - Clicking on its name (separate window with more details will be displayed)
 - Adding it to watch (left mouse button click on its name and select Add Watch Expression and then fill then name of the variable into “Add Watch Expression” window)

Monitoring of variables in debug mode

- We can set a breakpoint within HAL_ADC_ConvCpltCallback() which would be called once complete buffer is filled with data.



The screenshot shows the STM32 CubeIDE interface. On the left, the code editor displays the HAL_ADC_ConvCpltCallback function in stm32g0xx_hal_tim.c. A breakpoint is set at line 305, which contains the condition `if(HAL_ADC_Stop_DMA(&hadc1) != HAL_OK)`. On the right, the Expressions window shows the state of the ADC buffer. The buffer is of type `uint16_t [8]` and its address is `0x2000012c`. The values of the buffer elements are as follows:

Expression	Type	Value
ADC_buffer	uint16_t [8]	0x2000012c <ADC_buffer>
ADC_buffer[0]	uint16_t	919
ADC_buffer[1]	uint16_t	919
ADC_buffer[2]	uint16_t	921
ADC_buffer[3]	uint16_t	920
ADC_buffer[4]	uint16_t	920
ADC_buffer[5]	uint16_t	916
ADC_buffer[6]	uint16_t	919
ADC_buffer[7]	uint16_t	920


- Now we need to recalculate measured values into deg C values. We can use macro `__LL_ADC_CALC_TEMPERATURE()`

Thank you



 [/STM32](https://www.facebook.com/STM32)

 [@ST_World](https://twitter.com/ST_World)

 community.st.com