AN5816

Application note

How to build a LPBAM application on STM32U5 MCUs using STM32CubeMX

Introduction

This application note shows how to build an LPBAM (low-power background autonomous mode) application using the new STM32CubeMX LPBAM feature on STM32U5 MCUs. It details all the necessary steps and instructions to follow in STM32CubeMX, and the APIs that must be added to the STM32CubeMX (version 6.9.0) generated code.

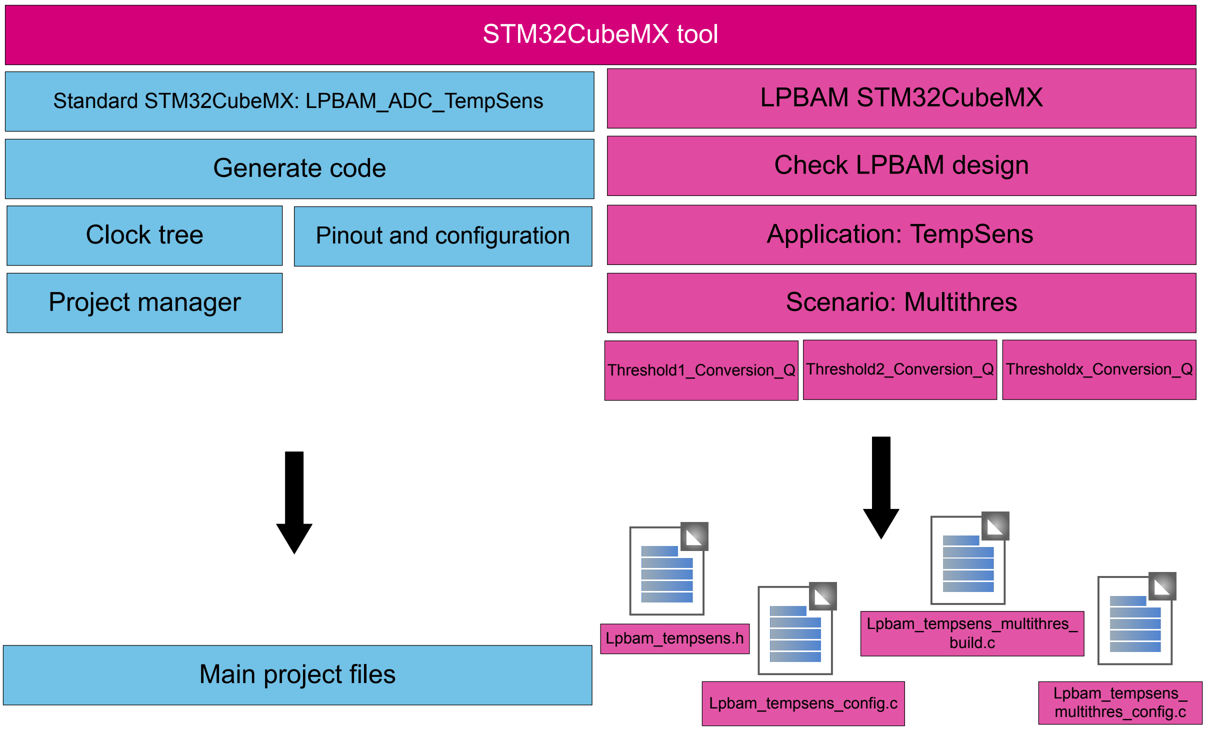
This document does not provide details about LPBAM operating mode, nor the LPBAM utility, nor about the STM32CubeMX LPBAM feature. It includes only the steps necessary to build the application.

1 General information

LPBAM is an operating mode that allows peripherals to function autonomously, independently of power modes, and without running any software. A hardware subsystem embedded in the STM32 microcontroller implements LPBAM.

To create an LPBAM application using the STM32CubeMX tool, the user needs the STM32CubeMX standard view. This sets up the main application and code generation, and the LPBAM view to build the LPBAM applications.

Figure 1. SM32CubeMX tool scope



This document applies to all STM32U5 devices. All these products are Arm®-based microcontrollers.

Note: For more information on LPBAM, refer to the application note STM32U5 series power optimization using LPBAM (AN5645).

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

2 LPBAM TempSense application description

2.1 Principle

The purpose of this application is to run periodic ADC conversions in STOP2 mode, with converted value threshold detection. Once threshold is reached, the conversion period is modified staying in STOP2 mode. The threshold value is also modified, and its interrupt is enabled to wake up from Stop mode when the ADC converted data reaches this second threshold value

After reset, the main application configures the system clock to the maximum device frequency. This increases the application performance. It then configures the system power supply on the SMPS regulator to reduce the device power consumption.

The main application then configures the resources required for the main application (ICACHE, GPIO, DMA, LEDs).

When all of the resources needed are configured and ready, the main application calls the LPBAM APIs to initialize, build, link, and start the scenario.

The application scenario description is as follows:

- The ADC peripheral is configured to convert the internal temperature sensor mapped to its temperature sensor channel. The conversion starts with a period of 100 ms. For this application, LPTIM PWM is the trigger signal. The transfer of converted data from ADC peripheral to SRAM peripheral is done by a DMA channel. The transferred data is stored in a buffer (Threshold1_Data_Buffer).
  The ADC analog watchdog monitors the internal temperature signal to be converted. At this point, the analog watchdog 1 threshold is configured to the threshold_1 value.
  The whole system then enters STOP2 mode.
- When the internal temperature reaches the analog watchdog 1 threshold_1 value, the ADC analog watchdog 1 generates its trigger event. It does not use an interrupt, in order to avoid the system wake-up. This signal propagates to another DMA channel that updates the ADC conversion period to 10 ms (LPTIM PWM signal period), changes the buffer conversion to a new buffer (Threshold2_Data_Buffer), and configures the analog watchdog 1 to the threshold_2 value.
- When the internal temperature reaches the analog watchdog 1 threshold_2 value, the ADC analog watchdog 1 generates an analog watchdog interrupt. This wakes up the whole system from STOP2 mode.
- After the wake-up from STOP2 mode, the main application calls the LPBAM APIs to stop, unlink, and deinitialize the scenario.

2.2 Implementation

The following resources are needed in order to implement the scenario:

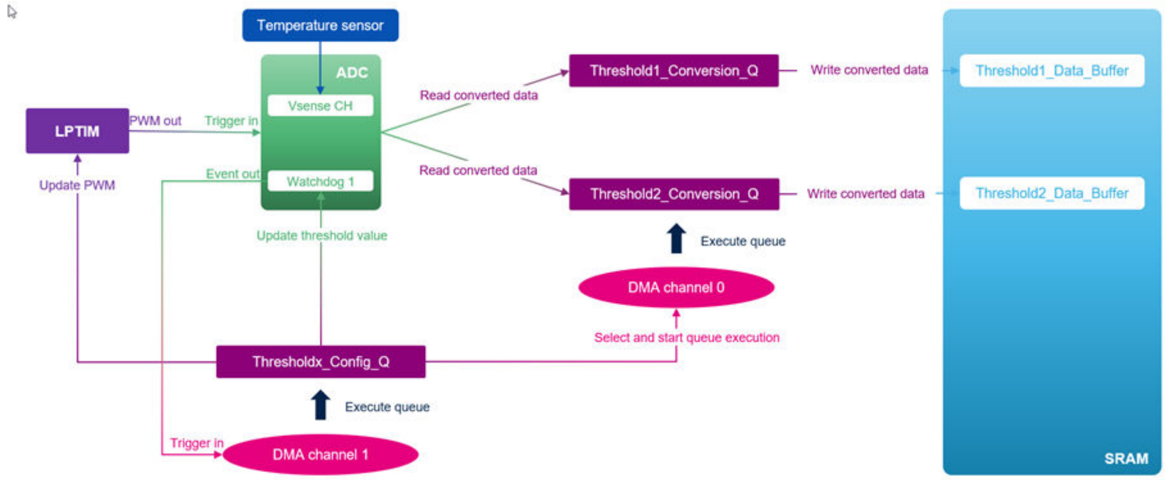
- an ADC peripheral with analog watchdog threshold-detection capability
- an LPTIM peripheral with PWM generation capability. Linked internally to ADC peripheral trigger input.
- a first DMA channel, to ensure the storage of converted data
- a second DMA channel to ensure:
  - starting of the first DMA channel
  - reconfiguration of the ADC analog watchdog 1 threshold
  - update of LPTIM period, and its reconfiguration
  - restart of the first DMA channel after detection of threshold 1.

To implement this scenario, the user must:

1. Configure the ADC peripheral (using the HAL), to perform conversions on the internal temperature sensor channel.
2. Configure the ADC analog watchdog 1 (using the HAL), to detect the temperature sensor threshold 1 value in silent mode (with no interrupt, to avoid system wake-up).
3. Configure the LPTIM peripheral (using the HAL) to generate a PWM signal with 100 ms period, and 50% duty cycle. This is used as a trigger source for the ADC.
4. Build an LPBAM queue 1, named Threshold1_Conversion_Q (with no DMA trigger condition). This allows:
   - ADC conversion start (with no configuration)
   - storage of converted data below threshold 1 in the SRAM Threshold1_Data_Buffer buffer.

5. Build an LPBAM queue 2 named Threshold2_Conversion_Q (with no DMA trigger condition). This allows:
   – ADC conversion start (with no configuration)
   – storage of converted data below threshold 2 in the SRAM Threshold2_Data_Buffer buffer.
6. Build an LPBAM queue 3 named Thresholdx_Config_Q (with analog watchdog 1 DMA trigger condition). This allows:
   – queue 1 execution start, at the first step
   – ADC conversion stop
   – LPTIM PWM period update
   – reconfiguration of the analog watchdog threshold 2 (with interrupt generation capability)
   – queue 2 execution start.

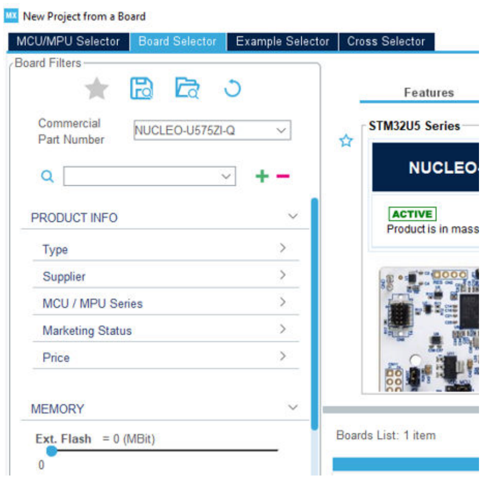
Figure 2. Block diagram



2.3 STM32CubeMX LPBAM TempSense application building

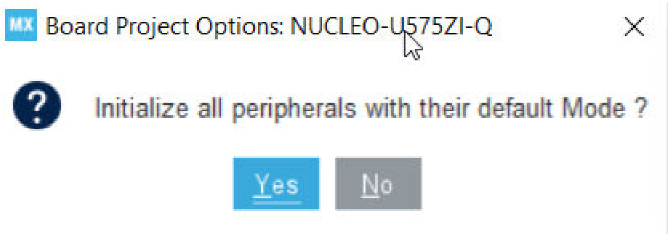
Open STM32CubeMX, version 6.9.0 is used to develop this application, and choose your board in the board selector menu.

Figure 3. NUCLEO-U575ZI-Q board selection



Then choose start project with no peripheral initialization pushing on "NO" button. (To avoid generation of useless project code, the required peripherals are initialized later.)

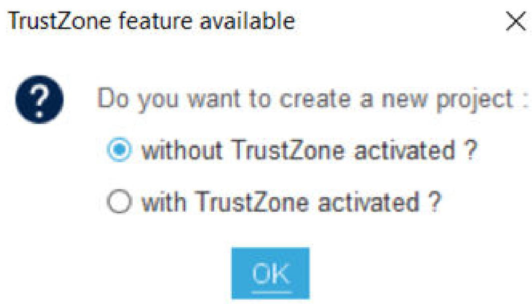
Figure 4. **Peripheral initialization**



To build an LPBAM application when TrustZone® is deactivated:

1. Choose "without TrustZone activated".
2. Click on OK.

Figure 5. **TrustZone® deactivated**

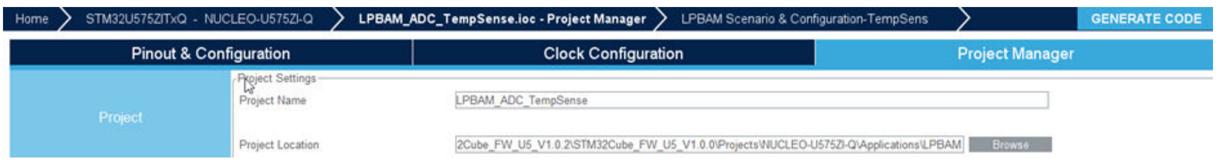


The STM32CubeMX tool entry point is always the standard view. Opening the project manager and saving the main project is recommended.

For this application:

1. Click on project manager panel.
2. Name the project LPBAM_ADC_TempSense
3. For the "Project Location", save the project with the other examples in the firmware package.

Figure 6. **Project settings**

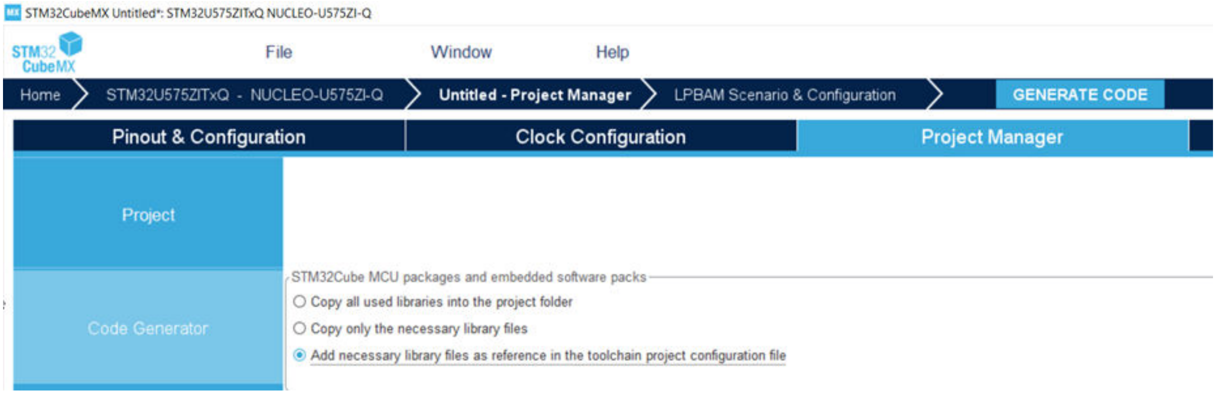


As this application is under the STM32Cube firmware file tree, there is no need to generate LPBAM utility files.

The user must then set up the code generator settings according to the targeted generated location:

1. Go to "Code Generator".
2. Choose "Add necessary library files as reference in the toolchain's project configuration file".

Figure 7. **STM32Cube MCU packages**



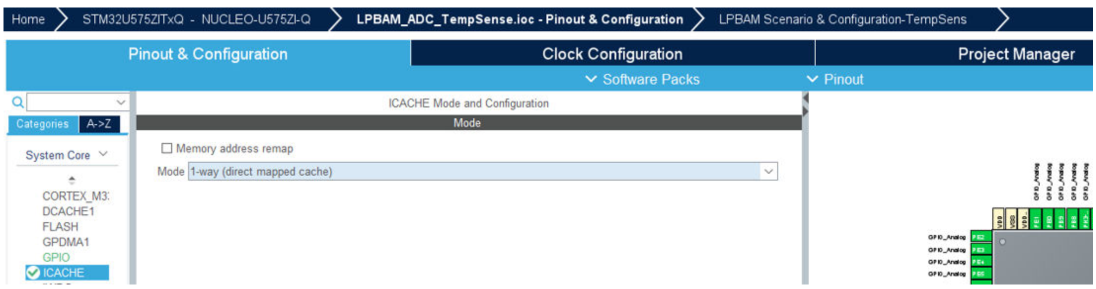
At this point, the project is configured and the *.IOC* file is saved under the selected path.

The user then opens the “Pinout & Configuration” view.

From this view, the user can configure all the main application resources. For this project, only the system resources need to be configured.

To increase the system performance, enabling the ICACHE peripheral in one-way configuration is recommended. To do this, click on the "System Core" menu, then on the"ICACHE" peripheral, and change "Mode" to "1-way (direct mapped cache)".

Figure 8. **ICACHE activation**



DT73277V1

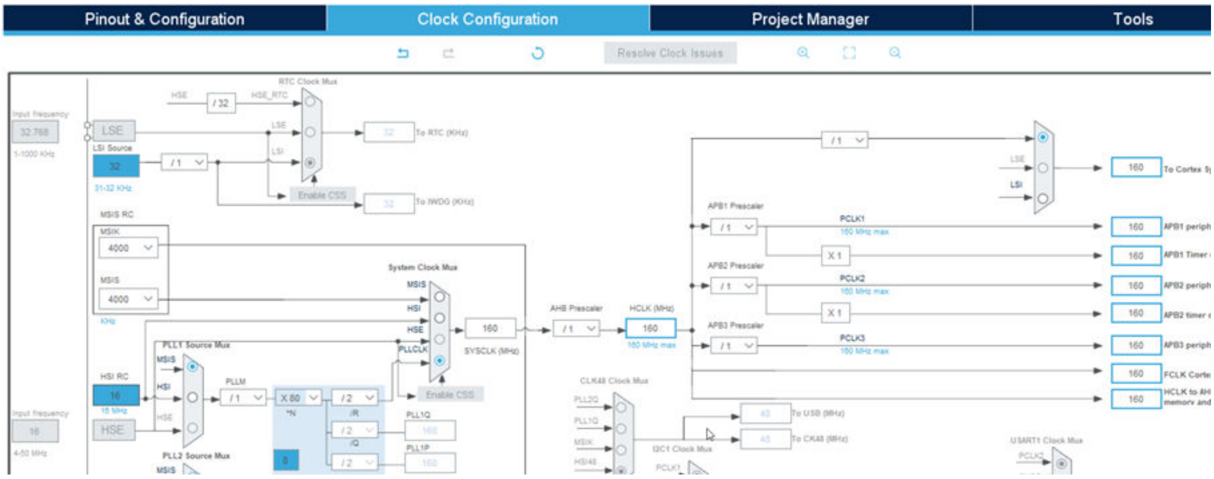
To reach the highest performance, configure the system clock to the highest value.

For this project, the user chooses PLL 1 as the system clock source. The PLL source is the MSI oscillator. (MSI range four is recommended to provide a 4 MHz clock signal).

The PLL 1 configuration is as follows:

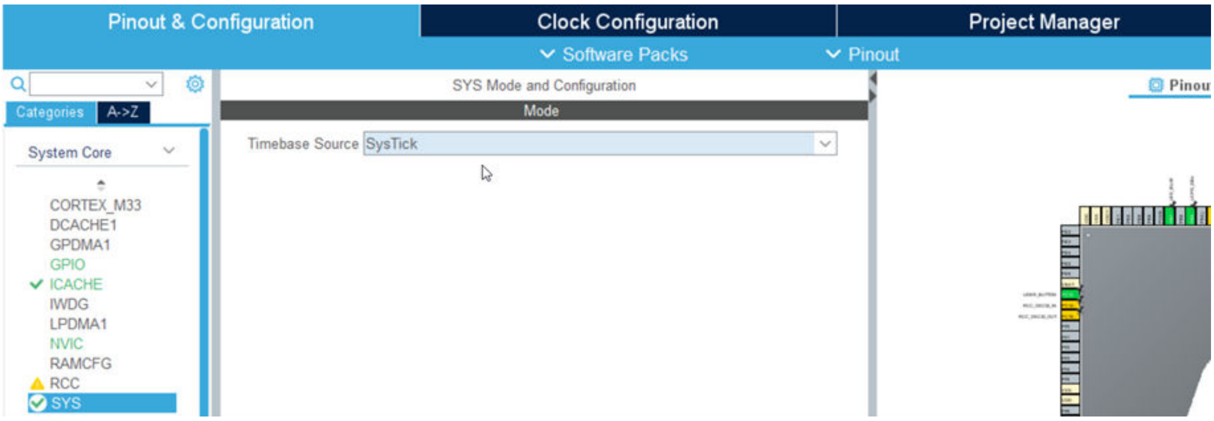
- PLLM = 1
- PLLN = 80
- PLLP = 2
- PLLQ = 2
- PLLR = 2

Figure 9. **Clock configuration**



For this project, the user chooses SysTick as the application timebase.

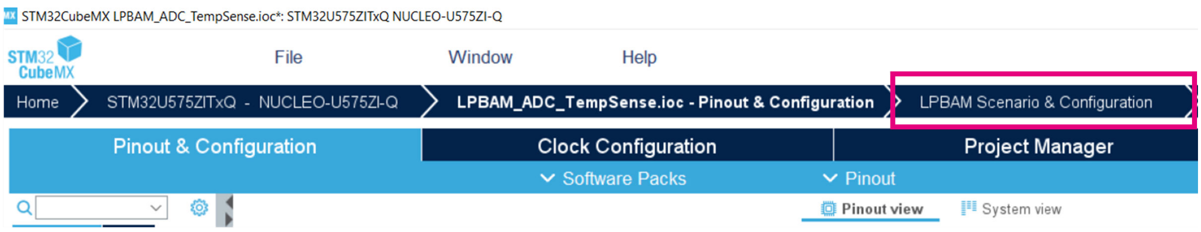
Figure 10. **Changing the timebase source**



At this point, the main project system is configured. On the next step, the user needs to build the LPBAM TempSense application.

Click on the "LPBAM Scenario & Configuration" panel.

Figure 11. **LPBAM scenario and configuration selection**



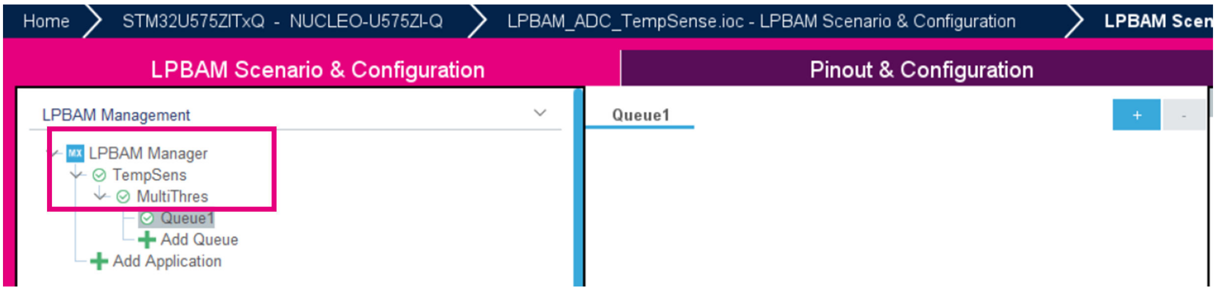
To add an LPBAM application, click on the “+ Add Application” option. This is situated on the left, under the LPBAM manager.

When adding an LPBAM application, the STM32CubeMX tool shows the LBPAM view. As for the standard view, it contains “LPBAM Scenario & Configuration”, “Pinout & Configuration” and “Clock Configuration” panels.

The naming chosen in the project should be reused in code-generated APIs and variables. This ensures consistency between STM32CubeMX LPBAM tool views and LPBAM generated application code. Carefully chosen user application naming is therefore recommended for clear and readable generated code. To do this:

1. Change the name of the application from LpbamAp1 to TempSens.
2. Change the name of the "Scenario" to "MultiThres".

Figure 12. Application and scenario naming

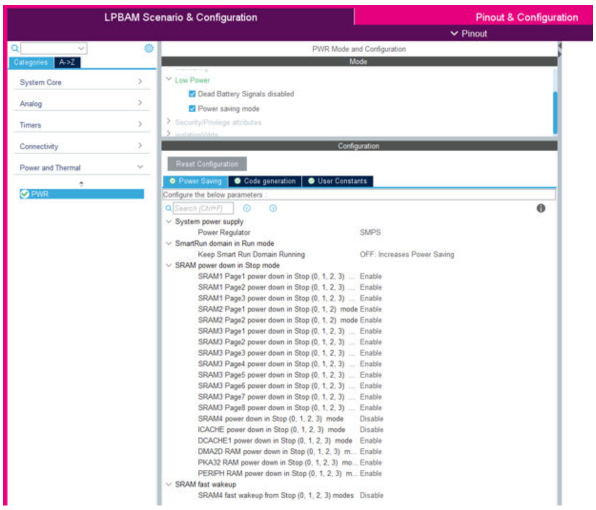


After renaming of the LPBAM application and scenario, the user needs to configure the system power.

To reach the lowest power consumption regarding the hardware target, the user shuts off all unused resources through the PWR peripheral. To do this:

- Click on the "Pinout & Configuration", then on "Power and Thermal", then on "PWR".
- Under "Low Power", select "Power saving mode", then enable the power-down for all the SRAMs, except SRAM4 and ICACHE.

Figure 13. PWR configuration



DT73278V1

At this point, the system is optimized in terms of consumption. It is ready to host the LPBAM application operation.

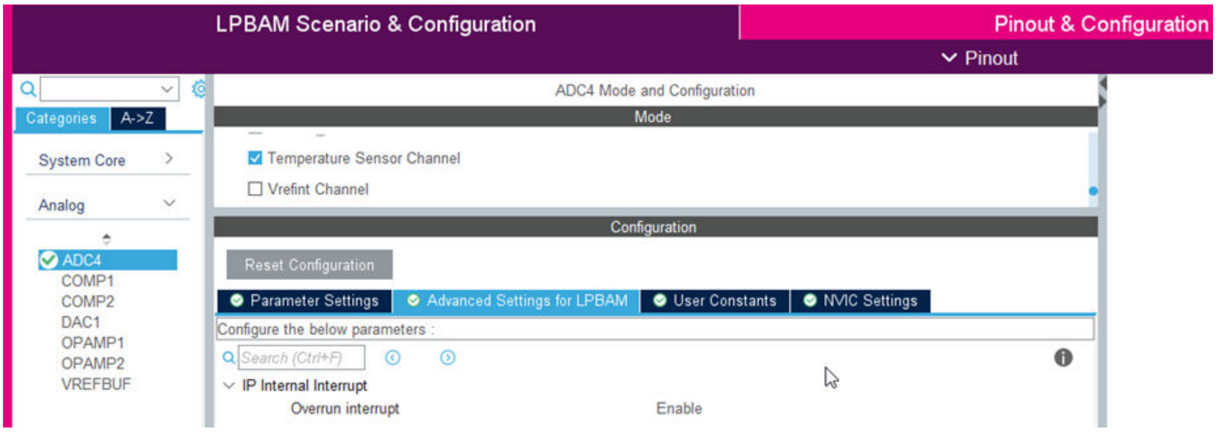
The user then needs to configure (using HAL drivers) the LPBAM scenario peripherals listed in Section 2.2: Implementation. To do this:

1. Click on "Analog", then on "ADC4".
2. An "ADC4 Mode and Configuration" window appears. Under "Mode", scroll down and select "Temperature Sensor Channel" (the only channel used in this application).

As any error occurring while in low-power mode stops the LPBAM mechanism running, it is recommended to enable any error interrupts.

Under the "Configuration" menu, click on "Advanced Settings for LPBAM" and enable "Overrun interrupt".

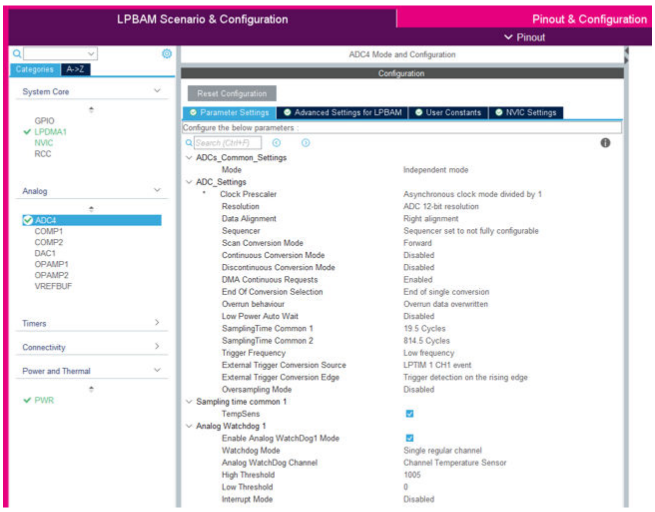
Figure 14. "ADC4 Mode" and "Advanced Settings for LPBAM" configuration



The ADC configuration sequence for this scenario is as follows:

1. Click on "Parameter Settings" and change the "Sequencer" to "Sequencer set to not fully configurable". (Use only internal ADC channel.)
2. Change the "Scan Conversion Mode" to "Forward".
3. Enable the "DMA Continuous Requests" (ADC conversions are performed in an infinite loop via DMA mode).
4. Change the "Sampling Time Common 1" to 19.5 cycles.
5. Change the "Sampling Time Common 2" to 814.5 cycles.
6. Change "External Trigger Conversion Source" to "LPTIM 1 CH1 event".
7. Enable "Analog WatchDog1".
8. Configure the "High Threshold" to 1005, which is equivalent to 0.809 V (1005 / 4095) * 3.3 V).

Figure 15. ADC4 parameter settings

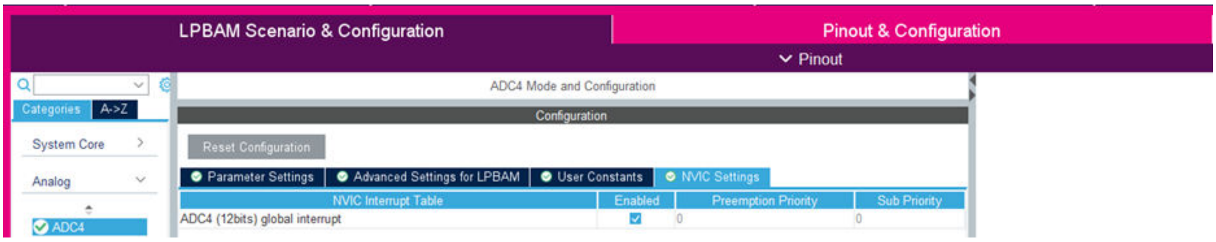


DT73279V1

The NVIC is used to detect ADC error interrupts, then wake the system from low-power mode. It is mandatory to enable this configuration via the NVIC settings.

Click on "NVIC Settings", and enable the "ADC4 global interrupt".

Figure 16. **ADC4 NVIC settings**

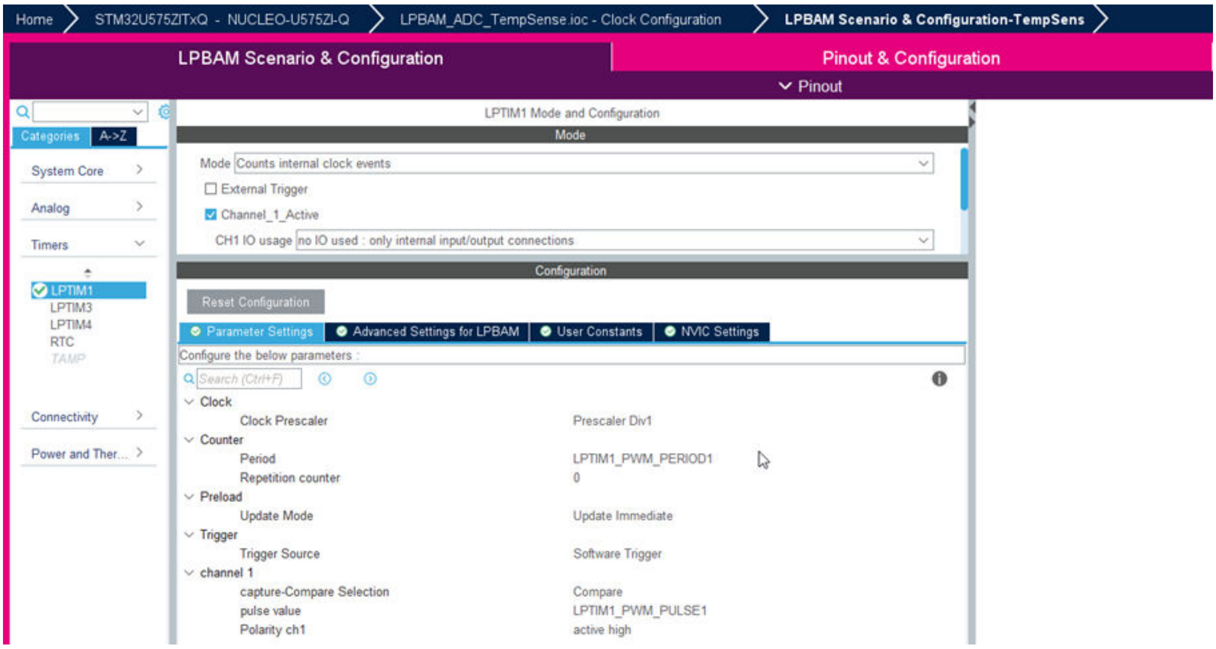


At this point, the ADC is configured and ready to start conversion, with analog watchdog monitoring enabled.

The LPTIM peripheral must then be configured to generate a PWM signal. It is set to 100 ms period, and 50% duty cycle:

1. Click on "Timers" then on "LPTIM1". The "LPTIM1 Mode and Configuration" window appears.
2. In "Mode", change the mode from "disable" to "Count internal clock events". Then select "Channel_1_Active".
3. Under "Configuration", in the "Parameter Settings":
   a. Click on , and select no check. Then change the "Period" to "LPTIM1_PWM_PERIOD1".
   b. Under "channel 1", change "capture-Compare Selection" from "Capture" to "Compare", and "pulse value" to "LPTIM1_PWM_PULSE1".

Figure 17. **LPTIM1 activation and parameter settings**



On application start up, the LPTIM period must be 100 ms and the LPTIM clock frequency is 32 kHz.

The auto-reload and counter compare values are calculated as follows:

$$Auto\_reload = (counter\_clock\_frequency \cdot period) - 1 = (32.10^3 \cdot 0.1) - 1 = 3199 \quad (1)$$

$$Counter\_compare = (counter\_clock\_frequency \cdot pulse) - 1 = (32.10^3 \cdot 0.05) - 1 = 1599 \quad (2)$$

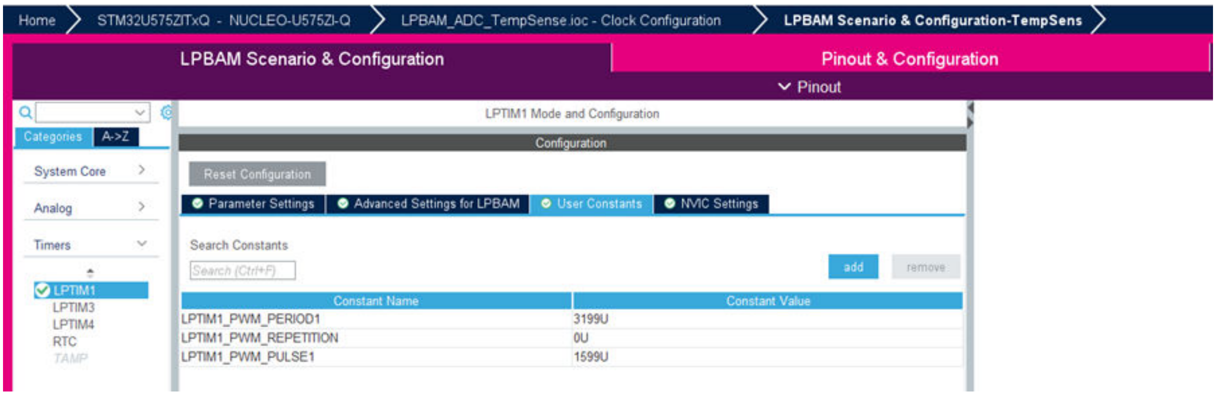
Where frequency is measured in Hz, and period and pulse are in seconds.

In the “User Constants” panel, click on "add "and create three "Constants":

- LPTIM1_PWM_PERIOD1 = 3199U
- LPTIM1_PWM_REPETITION = 0U

- LPTIM1_PWM_PULSE1 = 1599U

Figure 18. **User constants adding**



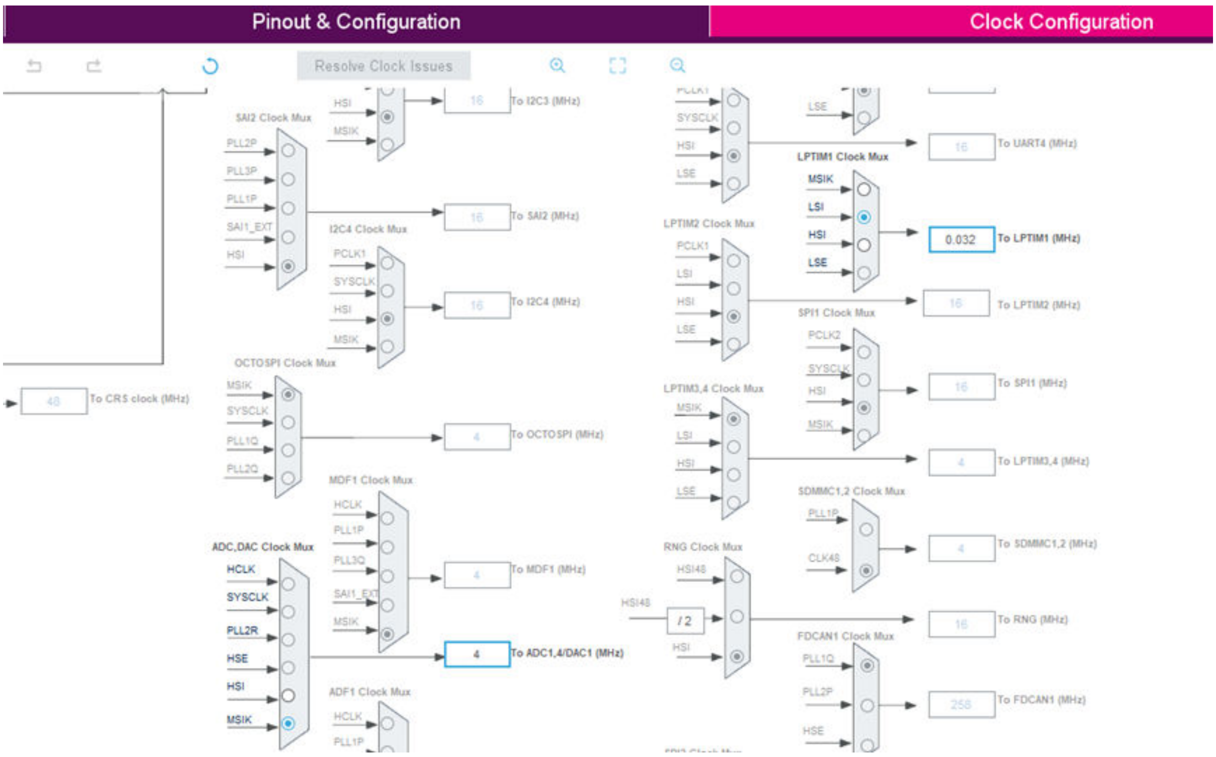
At this point, the ADC and LPTIM only need the LPTIM clock configuration before they can generate a PWM signal. The user must also configure the wake-up and peripheral kernel clocks. These allow the LPBAM application peripheral to operate in low-power mode.

For optimum power consumption, the wake-up clock frequency and kernel clocks sources should be chosen carefully.

When using the MSIK as kernel clock, use of the same scale (range 4 for this application) as an MSIK system clock is recommended:

1. Choose MSIK as ADC bus clock.
2. Choose LSI as LPTIM kernel clock .

Figure 19. **LPTIM1 and ADC4 clock configuration**



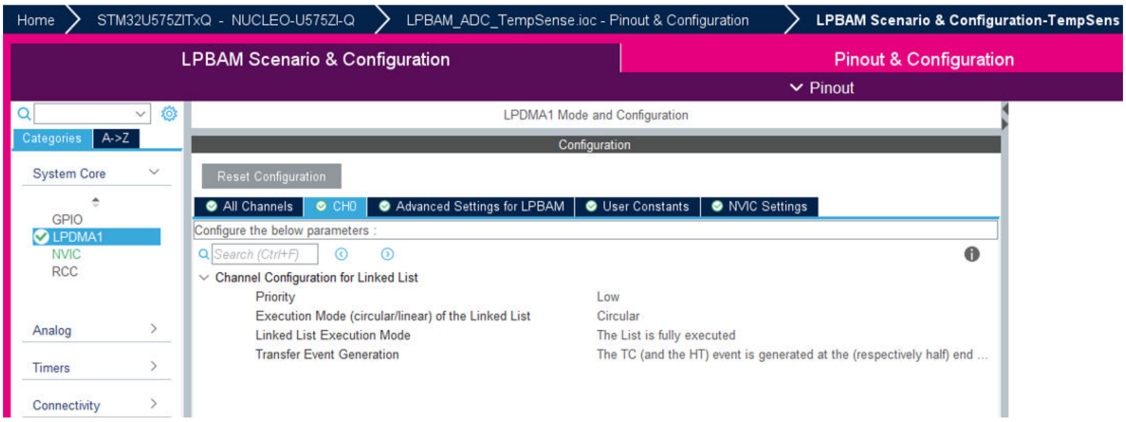
The next step is to configure the DMA channel that ensures the transfer of converted data from ADC to SRAM threshold buffers.

In LPBAM mode, the DMA channel must be configured in linked-list modes.

When DMA transfer is done in an infinite loop, the DMA channel execution mode should be configured in circular mode:

1. Left-click on "System core", then on "LPDMA1". Enable the "Linked-List Mode" for "CH0".
2. Click on "CH0". Then change the "Execution Mode" to "Circular" so that ADC conversion tasks are run in an infinite loop.

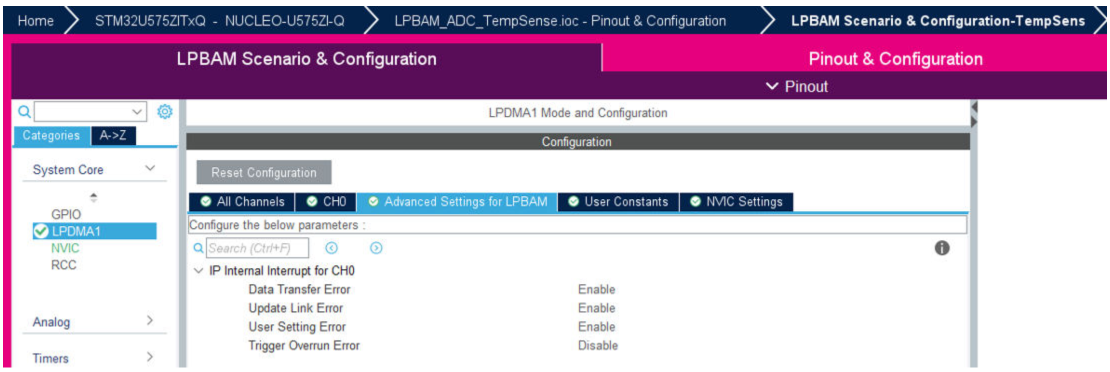
Figure 20. "LPDMA1 Mode" and "CH0" configuration



Click on the "Advanced Settings for LPBAM". Then enable:

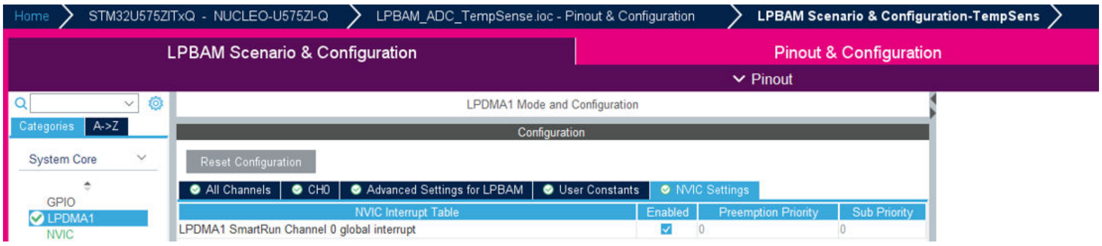
- "Data transfer Error"
- "Update Link Error"
- "User Setting Error"

Figure 21. "LPDMA1 Advanced Settings" for LPBAM configuration



Next, go to "NVIC Settings" and enable "LPDMA1 SmartRun Channel 0 global interrupt".

Figure 22. **LPDMA1 NVIC settings**



DT73282V1

All LPBAM peripherals are now configured and ready for scenario queue building.

It is mandatory to ensure that all used resources are configured before starting to build the scenarios.

2.4 Scenario queue building

As for the LPBAM application and scenario, the naming chosen for queues in the project is reused in code-generated APIs and variables.

Note: The chosen queue name is automatically suffixed with "_Q" in the generated code.

Now, go to the "LPBAM Scenario & Configuration" tab. Change the name of "Queue1" to "Threshold1_Conversion".

Figure 23. **Queue naming**

The user must then build the Threshold1_Conversion queue functionalities by using the LPBAM function toolbox library.

The function toolbox provides functionalities that perform diverse LPBAM scenarios for each supported peripheral.

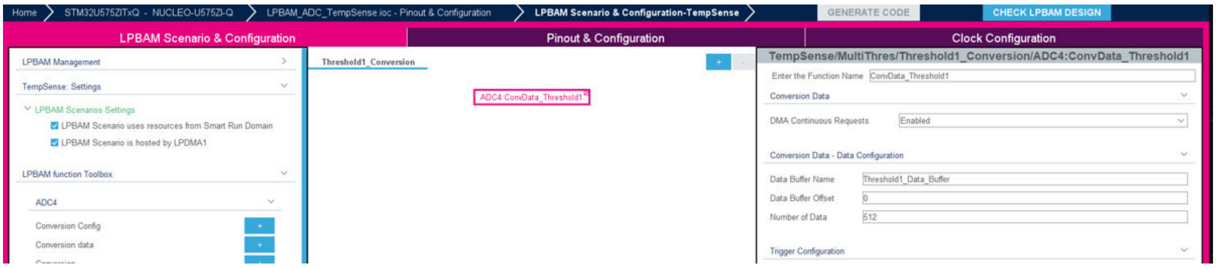
As for LPBAM queues, each LPBAM peripheral function can be renamed. By default, the STM32CubeMX LPBAM tool provides a default name. The function name is used in code-generated variable names.

The Threshold1_Conversion queue permits the ADC peripheral conversion and analog watchdog 1 monitoring to start. The converted data is stored in the Threshold1_Data_Buffer buffer area in SRAM. (The user does not need to reconfigure the ADC peripheral, as it is previously configured using the HAL.)

The ADC configuration for this queue is as follows:

1. Go to the left, to the LPBAM function toolbox. Click on ADC4, then on the plus in front of the "Conversion data". An ADC4: "Conversion_data" box appears in the middle. Then go to the right, and change the function name to "ConvData_Threshold1".
2. Enable DMA continuous requests.
3. Change the data buffer name to "Threshold1_Data_Buffer".
4. Change the value of "Data Buffer Offset" to 0, and "Number of Data" to 512.

Figure 24. Queue1 building

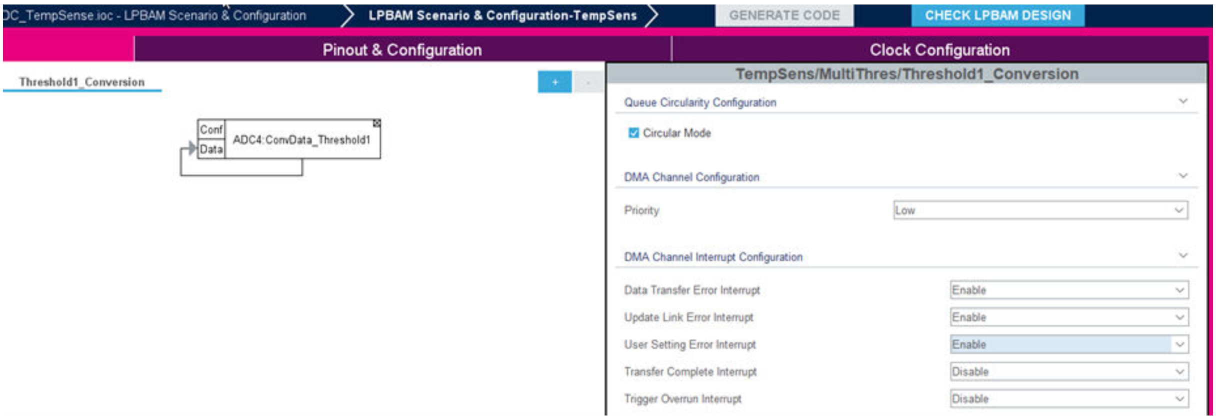


After adding all necessary functions to the active queue (respecting the order of function execution - first function, first execution), it is mandatory to configure the queue parameter.

For this queue, the ADC data conversion task must be done in an infinite loop. Also, the DMA channel that hosts this queue generates error interrupts to wake up the system from stop mode:

1. Go to the LPBAM manager and click on the "Threshold1_Conversion" queue.
2. Check "Circular Mode".
3. Enable: "Data Transfer Error Interrupt", "Update link Error interrupt", and "User Setting Error Interrupt".
4. Drag the arrow to face "Data" instead of "Conf", as only data conversion need to be repeated continuously.

Figure 25. Queue1 parameter configuration



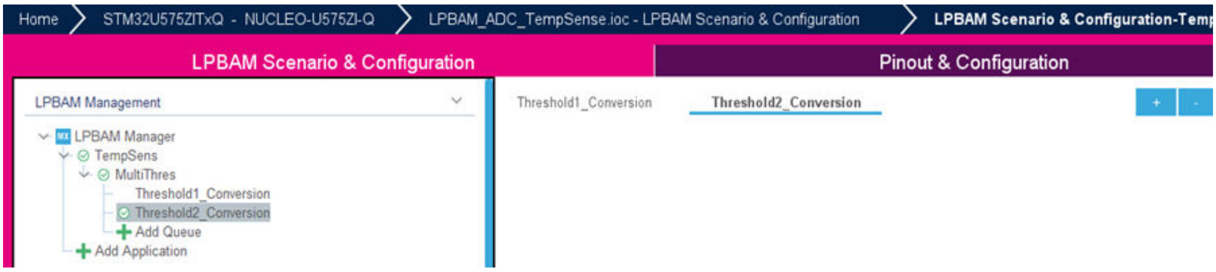
As for the Threshold1_Conversion queue, the Threshold2_Conversion queue allows:

- ADC peripheral conversion start
- analog watchdog 1 monitoring start
- data storage in the Threshold2_Data_Buffer buffer, and writing the data to SRAM

Note: The user does not need to reconfigure the ADC peripheral, as it is reconfigured with the Thresholdx_Config queue.

Click on "+Add Queue" to create a second queue, and name it "Threshold2_Conversion".

Figure 26. Queue2 naming



The ADC configuration for this queue is as follows:

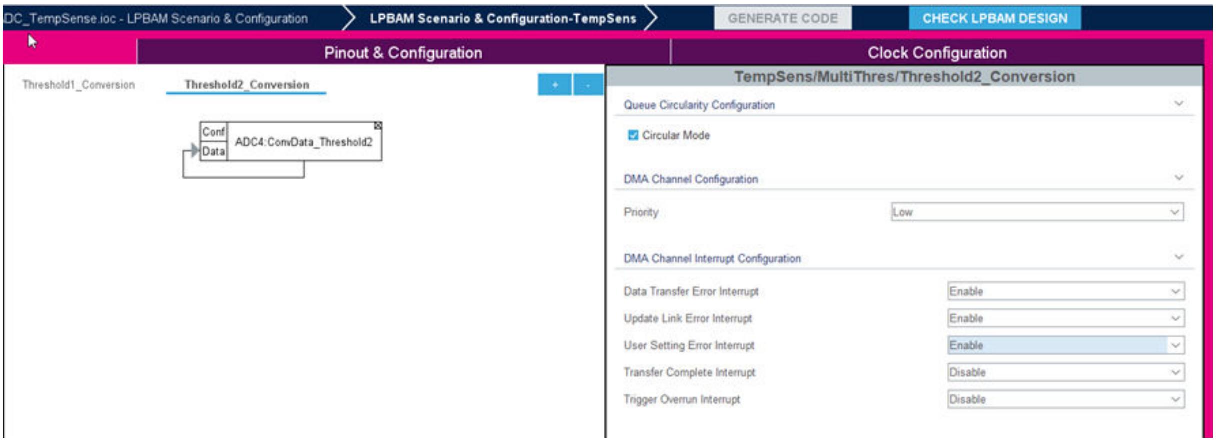
1. Go to the left, to the LPBAM function toolbox. Click on ADC4, then on the plus in front of "Conversion data".
2. Go to the right, and change the function name to ConvData_Threshold2.
3. Enable "DMA Continuous Requests".
4. Change "Data Buffer Name" to Threshold2_Data_Buffer.
5. Change the value of "Data Buffer Offset" to '0', and "Number of Data" to '512'.

Figure 27. Queue2 building

For this queue, the ADC data conversion task must be done in an infinite loop. Also, the DMA channel that hosts this queue generates error interrupts to wake up the system from stop mode in case an error has occurred.

- Go to the LPBAM manager and click on the "Threshold2_Conversion" queue.
- Check "Circular Mode".
- Enable "Data Transfer Error Interrupt", "Update Link Error Interrupt", and "User Setting Error Interrupt".
- Drag the arrow to face "Data" instead of "Conf", as only data conversion needs to be repeated continuously.

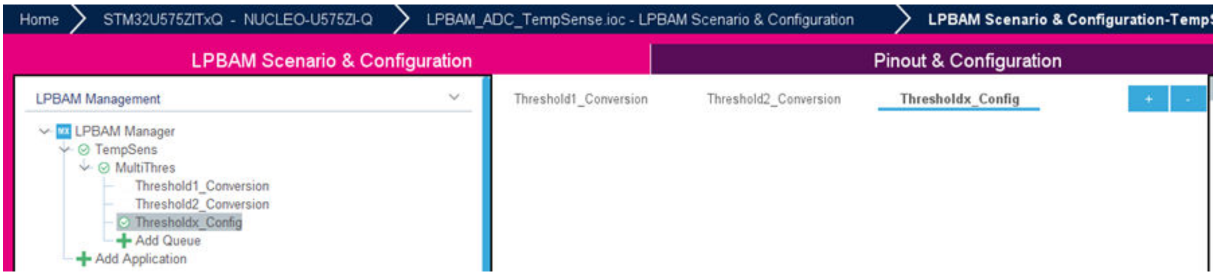
Figure 28. Queue2 parameter configuration



As for the Threshold1_Conversion and Threshold2_Conversion queues, the user must add, then rename the next queue to Thresholdx_Config.

Add a third queue, and name it "Thresholdx_Config".

Figure 29. **Queue3 naming**



The purpose of the Thresholdx_Config queue is to start the DMA channel execution of the Threshold1_Conversion queue when starting the LPBAM application execution.

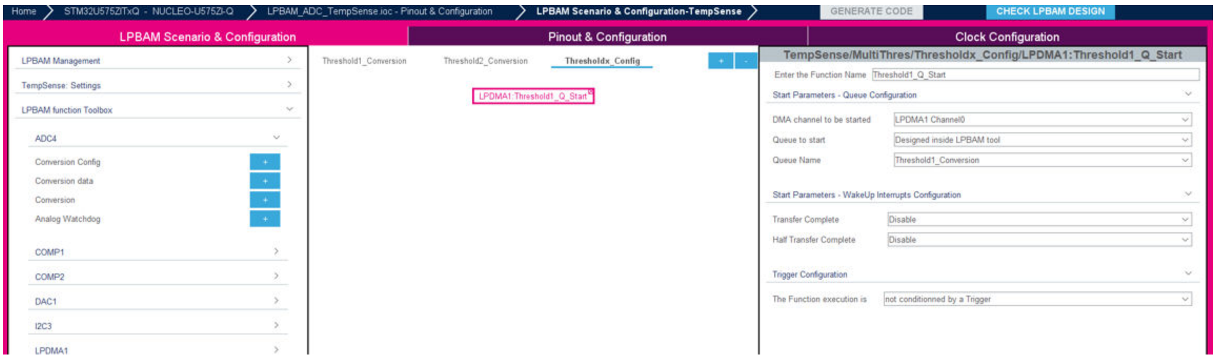
Detection of the analog watchdog 1 threshold 1 signal conditions the execution of the subsequent functionalities. When detected, the Thresholdx_Config queue should update the LPTIM PWM period value to 10 ms to increase ADC conversion cadence. Then reconfigure the analog watchdog threshold 2 value, and start the execution of the DMA channel that hosts the Threshold2_Conversion queue.

Unlike the other queues, the third queue has four nodes.

The LPDMA1 configuration sequence for the first node is as follows:

1. Go to the LPDMA1 and click on the "+" in front of start, then change the function name to "Threshold1_Q_Start".
2. Select the available DMA channel to execute the "Threshold1_Conversion" queue.
3. Change "Queue Name" from "No Selection" to "Threshold1_Conversion".

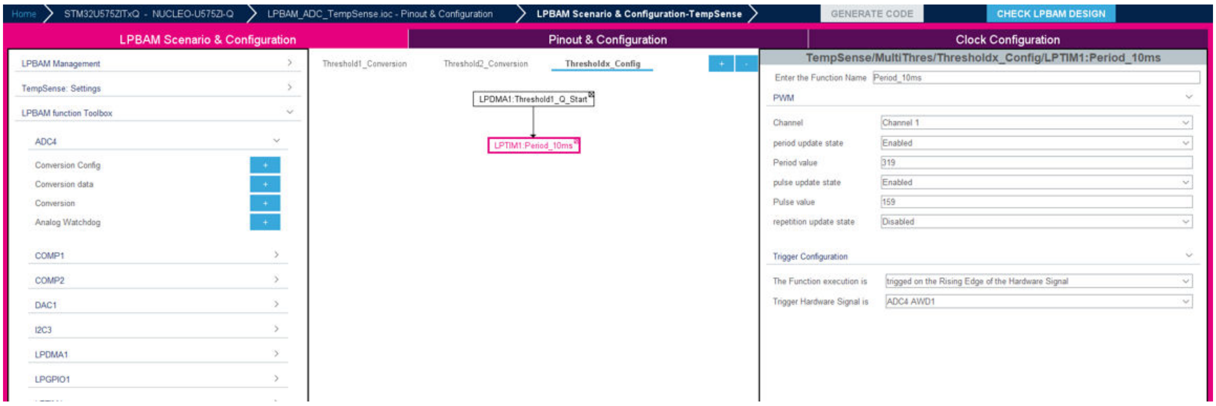
Figure 30. **First node configuration**



The LPTIM1 configuration sequence for the second node is as follows:

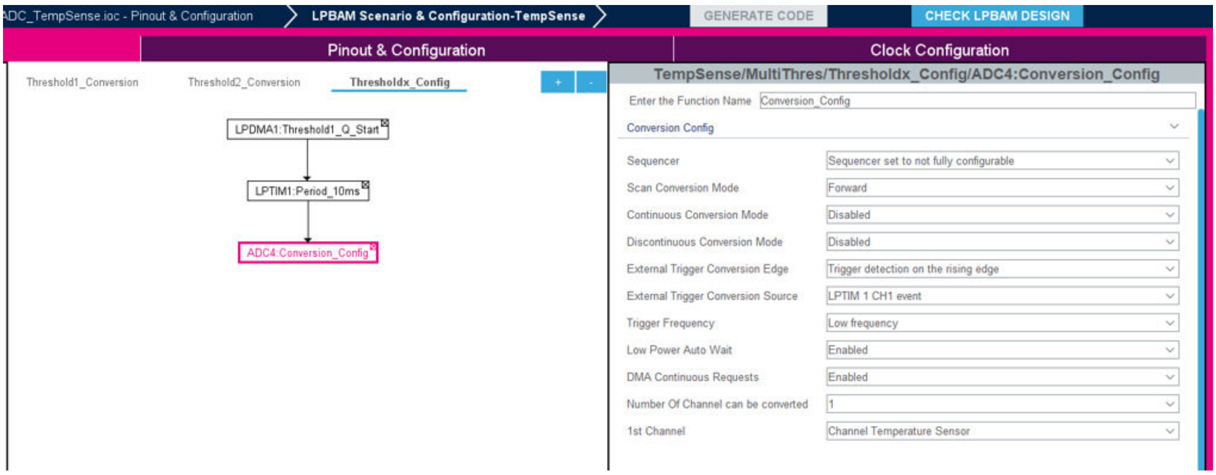
1. Go to LPTIM1 and click on the "+" in front of PWM.
2. Change the function name to "Period_10ms".
3. Enable the period update state, and insert 319 as the period value.
4. Enable the pulse update state, and insert 159 as the pulse value.
5. Under "Trigger Configuration", change "The Function execution is" from "Not conditioned by a trigger" to "Triggered on the Rising edge of the hardware Signal". This conditions the LPTIM update execution by analog watchdog 1 threshold 1 detection.
6. Change "Trigger hardware Signal is" from "EXTI line 0" to "ADC4 AWD1".

Figure 31. Second node configuration



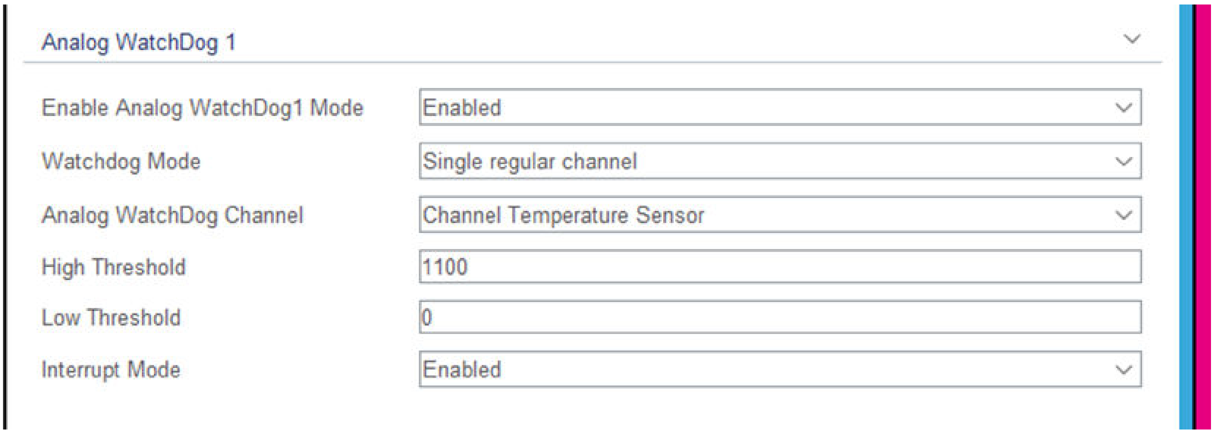
The ADC4 configuration for the third node is as follows:
- Click on ADC4, then on the "+" in front of the conversion config
- Change the name of the function to "Conversion_Config"
- Change "Sequencer" to "Sequencer set to not fully configurable"
- On the right, under the "Conversion_Config" disable the "Discontinuous Conversion Mode"
- Change "External Trigger Conversion edge" from "none trigger detection" to "Trigger detection on the rising edge"
- Enable "DMA Continuous Requests"
- Change "1st Channel" to " Temperature Sensor"

Figure 32. Third node conversion configuration



The analog watchdog configuration is as follows:
1. Enable "Analog WatchDog1"
2. Change "Analog WatchDog Channel" to "Channel Temperature Sensor"
3. Configure "High Threshold" to 1100
4. Enable "Interrupt Mode" to wake up the system after the detection of the analog watchdog 1 threshold 2 value.

Figure 33. **Third node "Analog Watchdog1" configuration**



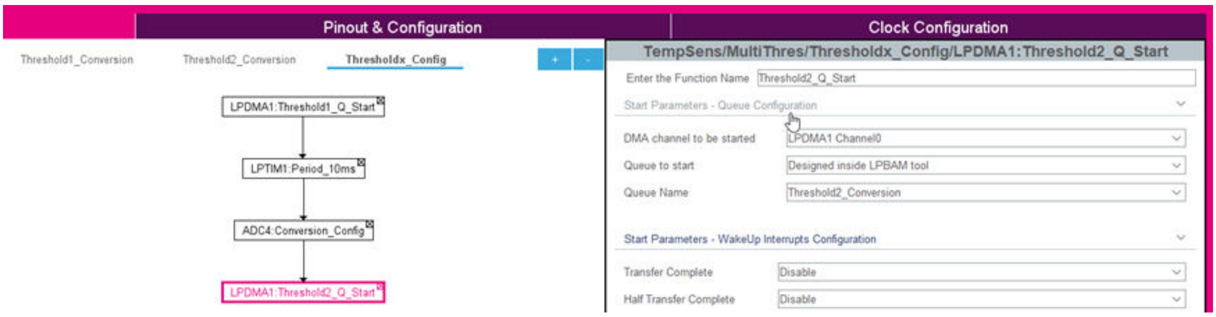
The LPDMA1 configuration for the fourth node is as follows:

1. Click on LPDMA1, then on the plus in front of start and change the function name to "Threshold2_Q_Start".
2. Select the available DMA channel to execute the Threshold2_Conversion queue.

Note: Use of the same DMA channel is recommended in this case. This reduces power consumption, as the Threshold1_Conversion and Threshold2_Conversion queues are never executed simultaneously.

3. Change the queue name to "Threshold2_Conversion".

Figure 34. **Fourth node configuration**

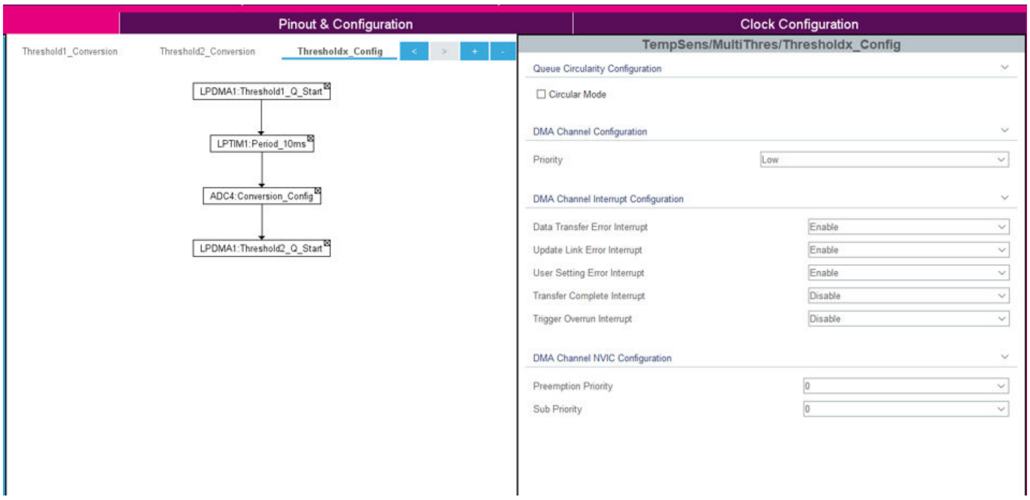


In this queue also, the user needs the DMA channel to generate error interrupts. These wake the system up from stop mode in case an error has occurred.

The third queue parameter configuration is as follows:

- Go to the left, and click on "Thresholdx_Config" under "Threshold2_Conversion"
- Enable "Data Transfer Error Interrupt", "Update Link Error Interrupt", and "User Setting Error Interrupt"

Figure 35. **Queue3 parameter configuration**



DT73283V1

At this point, the "LPBAM TempSense" application is built in the STM32CubeMX LPBAM tool.

It is then recommended to check the LPBAM design via the “Check LPBAM design” button. The STM32CubeMX tool checks, in the background, the consistency of the built application. It returns detected issues in the LPBAM log.

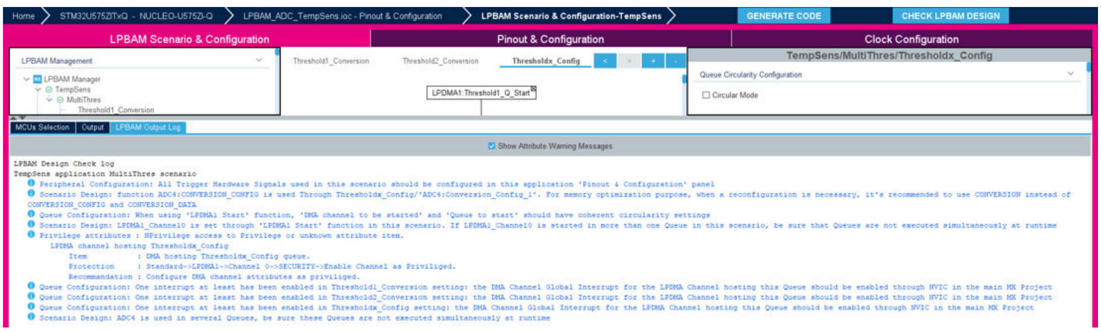
Figure 36. **LPBAM design check**

The LPBAM log returns general information to help in LPBAM scenario customization. This information is in accordance with the scenario-build information.

The check LPBAM design step is optional, but recommended.

If the LPBAM log is clean (no warning “Chestnut messages” or error “Pink messages”), the user can return to the STM32CubeMX standard view to generate code.

Figure 37. **LPBAM "Output Log"**



DT73284V1

Note: This is not the only way to make an LPBAM application. Using other steps can also work, but this is not recommended.

Click on "pinout" and "configuration" to return to the STM32CubeMX standard view.

Any LPBAM application made by the STM32CubeMX LPBAM tool is DMA channel agnostic. Therefore, the LPBAM application configures the DMA channel chosen to host the queue. It is up to the main application to allow the necessary DMA channels to host the LPBAM application.

For this application, a DMA channel hosts the LPBAM queue.

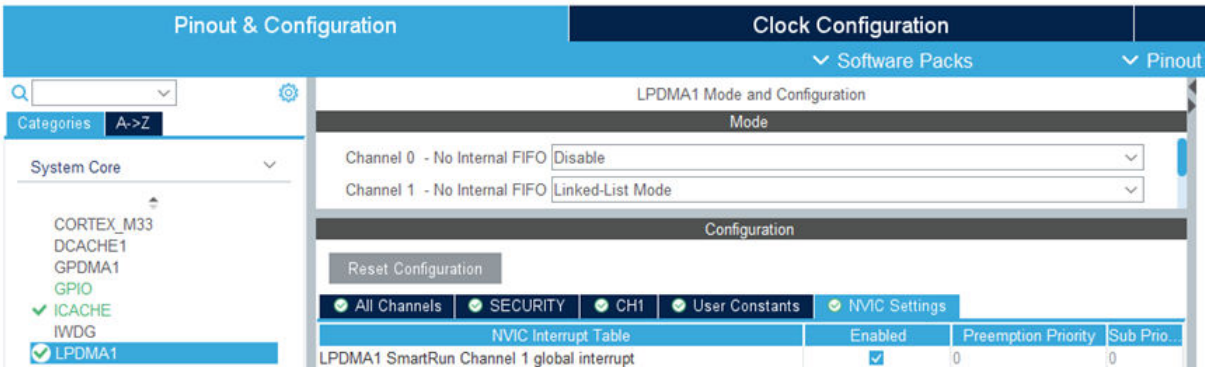
As DMA channel 0 is used to host the Threshold1_Conversion and Threshold2_Conversion queues, the user must choose any other DMA channel.

The LPBAM application configures the DMA channel chosen to host the queue. Hence, through the STM32CubeMX standard the user only needs to enable the DMA channel and its interrupt line, as follows:

1. Click on LPDMA1, and choose any mode for channel 1. (The LPBAM application configures the mode and parameters.)
2. Go to "NVIC Settings" and enable the interrupt line.

Note: The LPDMA is chosen because it is functional down to Stop 2 mode.

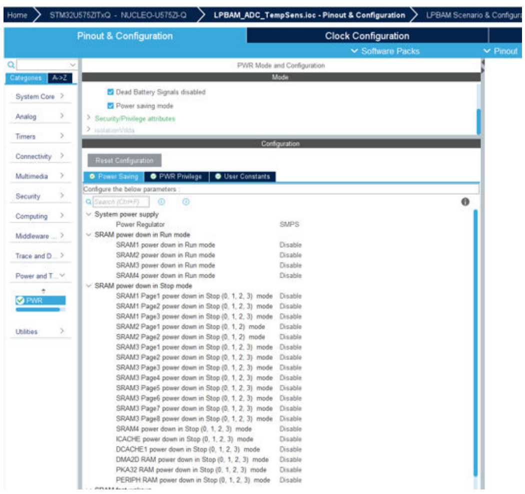
Figure 38. "LPDMA1 Mode" and "NVIC Settings" configuration



The user must configure the system power supply to achieve the lowest power consumption for the hardware target. The SMPS must be enabled through the PWR peripheral. To do this:

- Click on the "Pinout & Configuration", then on "Power and Thermal", then on "PWR".
- Under System power supply , select "SMPS" as power regulator.

Figure 39. Enable SMPS in main application



DT73285V1

At this point, all the LPBAM application needs are configured.

As the LPBAM targets the lowest possible power consumption, it is recommended that all unused pins are set to analog mode:

1. Click on "Pinout" and choose "Set unused GPIOs".
2. Set the highest number of GPIOs available (111 in this application).
3. Change "GPIO Type" from "Input" to "Analog".

Figure 40. Pinout configuration

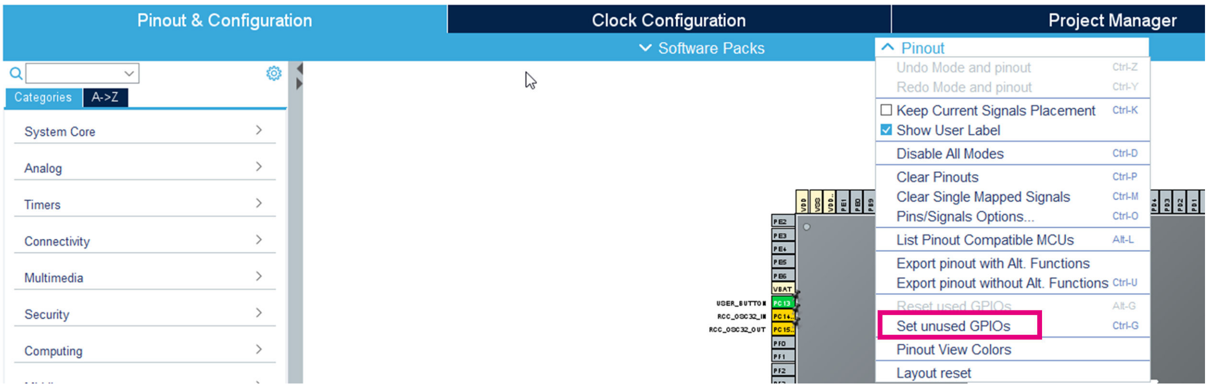
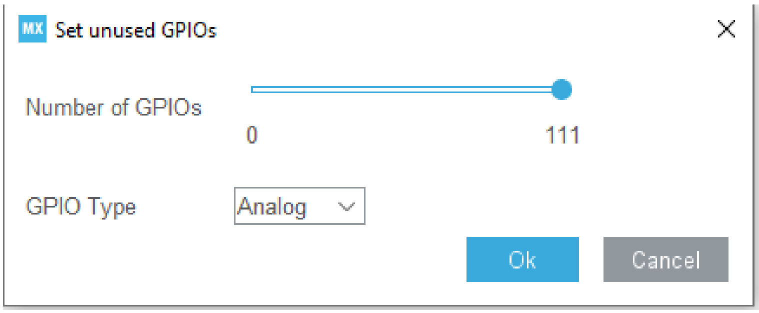


Figure 41. GPIO configuration



At this point, the main and the LPBAM applications are ready to be generated.

Click on "GENERATE CODE".

Figure 42. Code generation

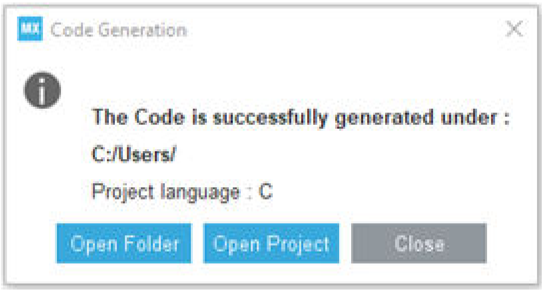
For the LPBAM application, the generated files are:

- lpbam_tempsens.h
- lpbam_tempsens_config.c
- lpbam_tempsens_multithres_build.c
- lpbam_tempsens_multithres_config.c

They contain all the APIs needed by the main application to manage any LPBAM application correctly.

After successful generation of the project, click on "Open Project".

Figure 43. **Code successfully generated**



Note: To use the BSP drivers, the user needs to add the necessary files, which are taken into account during the compilation.

2.5 How to call APIs

According to the project needs, the user has to call the necessary generated APIs.

This application uses the BSP library, so the user must include board resources in the main application.

Go to the main.h and add «*#include "stm32u5xx_nucleo.h"*» in the user section named “USER CODE BEGIN Includes”.

```
/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __MAIN_H
#define __MAIN_H
#ifdef __cplusplus
extern "C" {
#endif
/* Includes ------------------------------------------------------------------*/
#include "stm32u5xx_hal.h"
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stm32u5xx_nucleo.h"
/* USER CODE END Includes */
/* Exported types ------------------------------------------------------------*/
```

The main application can call the generated LPBAM application from any application file.

The LPBAM application is included in the main.c file.

In the main.c file, add «*#include "lpbam_tempsens.h"*» in the “USER CODE BEGIN Includes” user section.

```
/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "lpbam_tempsens.h"
/* USER CODE END Includes *
```

The main application must declare any buffers used in the LPBAM application. These are then exported to the LPBAM application files.

Add the buffer declarations to the “USER CODE BEGIN P” user section in main.c.

```
LPTIM_HandleTypeDef hlptim1;
/* USER CODE BEGIN PV */

/* Buffers declaration */
uint16_t Threshold1_Data_Buffer[512] = {0U};
uint16_t Threshold2_Data_Buffer[512] = {0U};

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
```

Any additional service can be added under user codes in the main.c file.

For this application, an application API is needed that:

- Enters Stop2 mode and checks whether or not the system was in stop mode
- Checks the content of the ADC buffers

These functionalities are added to the “USER CODE BEGIN 4” user section.

```
/* @brief Enter Stop2 mode and checks whether the system was in Stop2 or not.
 * @param None
 * @retval None
 */
static void Enter_Stop2_Mode(void)
{
  /* Enter the system to STOP2 mode */
  __HAL_RCC_PWR_CLK_ENABLE();
  HAL_PWREx_EnterSTOP2Mode(PWR_STOPENTRY_WFI);
  /* Check that the system was resumed from stop 2 */
  if (__HAL_PWR_GET_FLAG(PWR_FLAG_STOPF) == 0U)
  {
    Error_Handler();
  }
  /* Clear stop flag */
  __HAL_PWR_CLEAR_FLAG(PWR_FLAG_STOPF);
  /* Check that stop flag is cleared */
  if (__HAL_PWR_GET_FLAG(PWR_FLAG_STOPF) != 0U)
  {
    Error_Handler();
  }
}
/**
  * @brief Verify whether the buffer has a value greater than the threshold or not.
  * @param Buffer     Pointer to the buffer that contains the data to be checked.
  * @param BufferSize Size of the buffer to be checked.
  * @param Threshold The Threshold value.
  * @retval Comparaison status.
  */
static uint32_t ADC_Buffer_Check(void* Buffer, uint32_t BufferSize, uint16_t Threshold)
{
  uint32_t count = 0;
  /* Repeat for all buffer size */
  while (count < BufferSize)
  {
    /* Check if the buffer value is greater than threshold */
    if ( *(uint16_t*)((uint16_t*)Buffer + count) > Threshold )
    {
      return 1;
    }
    count++;
  }
  return 0;
}
```

Add the functions declarations in the main.c under the user section named “USER CODE BEGIN PFP”.

```
  /* USER CODE BEGIN PFP */
static void Enter_Stop2_Mode(void);
static uint32_t ADC_Buffer_Check(void* Buffer, uint32_t BufferSize, uint16_t ThresholdValue);
/* USER CODE END PFP */
```

To reach the lowest power consumption, all pins unused by the LPBAM scenario are configured in analog mode, including the debug pins. The user can use LEDs to observe the project runtime status.

In the core of the main() API, the LEDs must be initialized using the BSP layer inside the “USER CODE BEGIN 2” user section.

```
  /* USER CODE BEGIN 2 */
  /* Initialize LED1 and LED3 : GREEN and RED leds */
  BSP_LED_Init(LED1);
  BSP_LED_Init(LED3);
```

After initializing the LEDs, the user calls the LPBAM generated APIs to initialize, build, link, and start the LPBAM application within the “USER CODE BEGIN 2” user section.

The call sequence of the LPBAM application-generated APIs is unique. It is mandatory to follow this to make any LPBAM application run.

```
/* LPBAM TempSens application init */
MX_TempSens_Init();

/* LPBAM TempSens application MultiThres scenario init */
MX_TempSens_MultiThres_Init();

/* LPBAM TempSens application MultiThres scenario build */
MX_TempSens_MultiThres_Build();


/* LPBAM TempSens application MultiThres scenario link */
MX_TempSens_MultiThres_Link(&handle_LPDMA1_Channel1);

/* LPBAM TempSens application MultiThres scenario start */
MX_TempSens_MultiThres_Start(&handle_LPDMA1_Channel1);
```

At this point, the LPBAM application is operating. The lowest-power mode must then be entered to ensure the functional aspects.

For this application and device, Stop2 is the lowest-power mode.

```
/* Enter Stop2 mode */
Enter_Stop2_Mode();
```

At this point, the LPBAM application is still functional, while reaching the lowest power consumption. The power consumption is measured during this step. See detail in Section 3: Power consumption measurement .

After the detection of the low thresholds, the ADC analog watchdog generates its interrupt to allow the system to exit Stop2 mode. The LPBAM application is still running, as it is not affected by entering and exiting low power mode.

For this application, the user calls the following API in the main application's “USER CODE BEGIN 2” user section. This stops, unlinks and de-initializes the LPBAM application.

```
/* LPBAM TempSens application MultiThres scenario stop */
MX_TempSens_MultiThres_Stop(&handle_LPDMA1_Channel1);
/* LPBAM TempSens application MultiThres scenario unlink */
MX_TempSens_MultiThres_UnLink(&handle_LPDMA1_Channel1);
/* LPBAM TempSens application MultiThres scenario de-init */
MX_TempSens_MultiThres_DeInit();
```

At this point, all used resources are deinitialized and can be reused cleanly in the main application.

To ensure that the ADC analog watchdog thresholds are changed correctly, a check of buffer data converted is implemented for each threshold buffer. If any error is detected during this step, the applicative code calls the error handler API.

```
 /* Check that the ADC_Buffer contains a value greater than the threshold1 */
if (ADC_Buffer_Check(Threshold1_Data_Buffer, 512, 1005) == 0U)
{
  Error_Handler();
}
/* Check that the ADC_Buffer contains a value greater than the threshold2 */
if (ADC_Buffer_Check(Threshold2_Data_Buffer, 512, 1100) == 0U)
{
  Error_Handler();
}
```

Inside the error handler, the application turns on the red LED. This is implemented under the “USER CODE BEGIN Error_Handler_Debug” user section.

```
/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  /* Turn LED3 on */
  BSP_LED_On(LED3);
  while (1)
```

```
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
```

At this point, the main application is ready to manage the LPBAM application and its execution state (fail or success).

It is up to the user to export the main application user buffers, in the lpbam_tempsens_multithres_build.c file. For this, the buffer declaration is added under the “USER CODE BEGIN EV” user section.

```
/* External variables ----------------------------------------------------------------------
---------*/
/* USER CODE BEGIN EV */
/* LPBAM user buffers declaration */
extern uint16_t Threshold1_Data_Buffer[512];
extern uint16_t Threshold2_Data_Buffer[512];
/* USER CODE END EV */
```

The trigger signal management depends on the user application. The user must therefore add starting and stopping trigger signal APIs.

For this application, the ADCs LPTIM PWM signal is the conversion trigger.

Starting PWM signal generation is done by a call from the lpbam_tempsens_multithres_config.c file, under the “USER CODE BEGIN TempSens_MultiThres_Start” user section.

```
  /* USER CODE BEGIN TempSens_MultiThres_Start */
  /* LPBAM LPTIM1 start PWM generation */
  if (HAL_LPTIM_PWM_Start(&hlptim1, LPTIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE END TempSens_MultiThres_Start */
```

Stopping PWM signal generation is done by a call from the lpbam_tempsens_multithres_config.c file, under the “USER CODE BEGIN TempSens_MultiThres_Stop” user section.

```
 /* USER CODE BEGIN TempSens_MultiThres_Stop */
 /* LPBAM LPTIM1 stop PWM generation */
 if (HAL_LPTIM_PWM_Stop(&hlptim1, LPTIM_CHANNEL_1) != HAL_OK)
 {
 Error_Handler();
 }
 /* USER CODE END TempSens_MultiThres_Stop */
```

The ADC calibration depends on the device and application needs.

The user may add the calibration call (if needed) under the “USER CODE BEGIN ADC4_Init Calibration” user section.

```
 /* USER CODE BEGIN ADC4_Init Calibration */
 /*
 * ADC4 Calibration
 */
 if (HAL_ADCEx_Calibration_Start(&hadc4, ADC_CALIB_OFFSET, ADC_SINGLE_ENDED) != HAL_OK)
 {
 Error_Handler();
 }
 /* USER CODE END ADC4_Init Calibration */
```

For the ADC peripheral used by this LPBAM application, it is mandatory to add the enable of V_{DDA} power supply under the “USER CODE BEGIN ADC4_MspInit 0” user section.

```
 /* USER CODE BEGIN ADC4_MspInit 0 */
 /* Enable VDDA supply for ADC */
 HAL_PWREx_EnableVddA();
 /* USER CODE END ADC4_MspInit 0 */
```

For each interrupt callback generated in the lpbam_tempsens_multithres_config.c file, it is advisable to introduce code to report the main-application status. This code should be added under the callback user section.

For this application, the error (red) LED is added under the “USER CODE BEGIN Thresholdx_Config_DMA_Error_Callback” user section.

```
/**
 * @brief Thresholdx_Config queue dma error callback
 * @retval None
 */
static void MX_Thresholdx_Config_Q_DMA_Error_Callback(DMA_HandleTypeDef *hdma)
{
 /* USER CODE BEGIN Thresholdx_Config_DMA_Error_Callback */
 /* Turn on LED3 */
 BSP_LED_On(LED3);
 /* USER CODE END Thresholdx_Config_DMA_Error_Callback */
}
```

At this point, the project is ready and finalized. It can run safely with reporting any issues during run and low power modes.

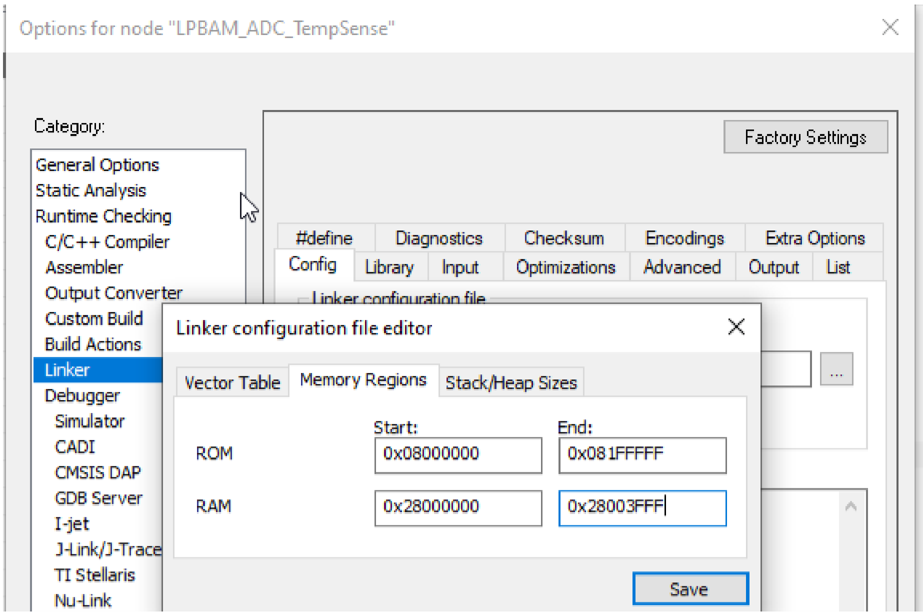
As the DMA access depends on device SOC integration, it is mandatory to check the accessibility of the DMA channel instance. This impacts the operating aspects of the LPBAM application.

For this application, the DMA channel used can access only SRAM4 in the same power domain.

According to the preferred IDE supported by this application, in the IAR project under files:

1. Right click on the LPBAM_ADC_TempSense
2. Click on options → "Linker" → "Memory Regions"
3. Change the start address of the RAM to 0x28000000 and the end address to 0x28003FFF (range of address accessible by the DMA channel)

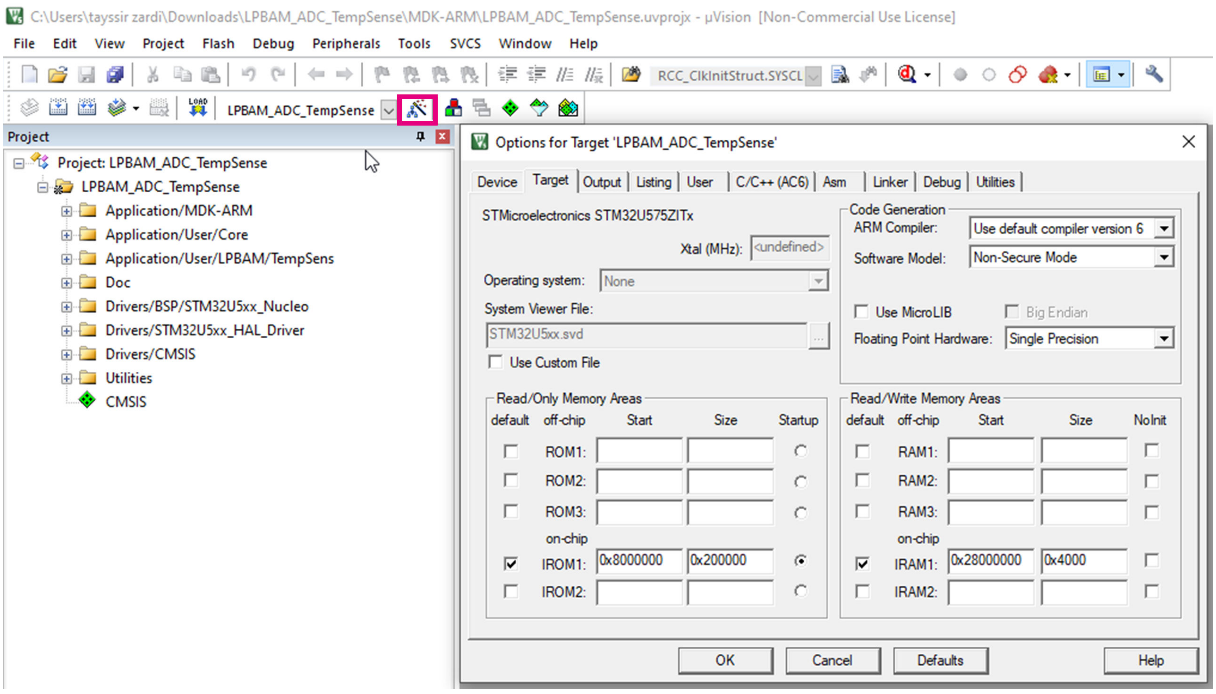
Figure 44. "Memory Regions" modification in IAR



In the Keil® project:

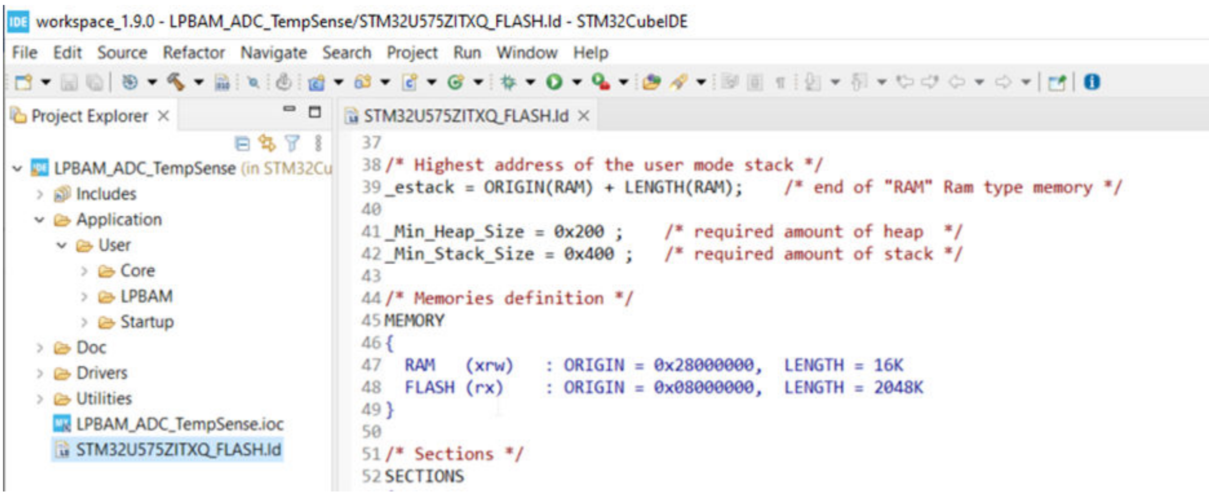
1. Open "Options" → "Target"
2. Change the start of the IRAM1 to 0X28000000 and the size to 0x4000

Figure 45. "Memory Regions" modification in Keil®



- In the STM32CubeIDE go to the STM32U575ZITXQ_FLASH.ld and configure the RAM ORIGIN = 0x28000000 and the LENGTH = 16 Kbytes

Figure 46. “Memory Regions” modification in the STM32CubeIDE



Placing the “RAM” section in the SRAM accessible by the DMA channel, is not the only way to make the LPBAM application functional. It is, however, the simplest method.

3 Power consumption measurement

To measure the power consumption of the application, the STM32 Power Shield application is used to supply the Nucleo board.

PowerShield is a plug-and-play solution intended to ease power consumption measurements.

Figure 47. Current consumption signal during project execution

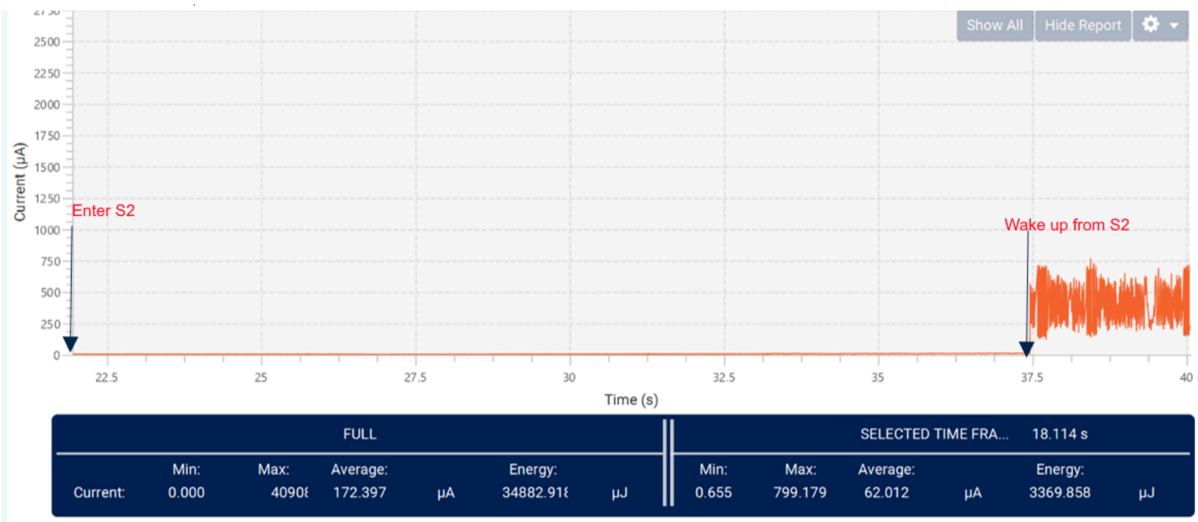
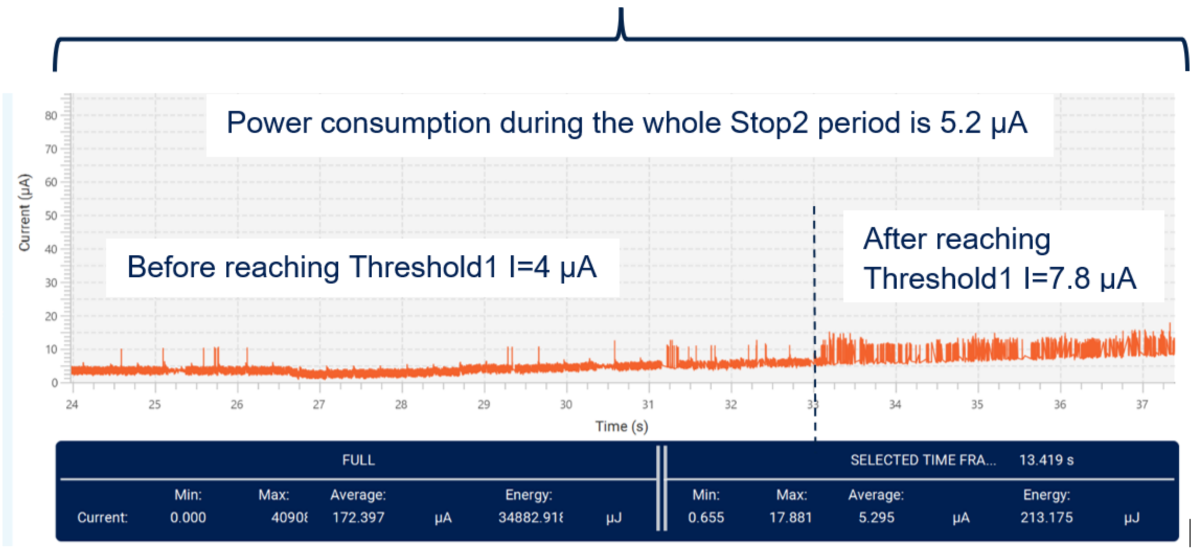


Figure 48. Current consumption signal during stop 2 mode



Revision history

Table 1. Document revision history

Date	Version	Changes
03-Oct-2022	1	Initial release.
14-Dec-2022	2	Updated: • Introduction • Section 1 General information • Section 2.3 STM32CubeMX LPBAM TempSense application building
23-Aug-2023	3	Updated: • Introduction • STM32CubeMX LPBAM TempSense application building • Scenario queue building • How to call APIs
07-Feb-2024	4	Title updated

Contents

List of tables

List of figures