# How to implement advanced power management on STM32H747/757 MCUs

## Introduction

The STM32H747/757 microcontroller line is based on the high-performance Arm® Cortex®-M7 and Cortex®-M4 32-bit RISC cores. The system is partitioned into three power domains that operate independently. This allows user applications to obtain the best trade-off between power consumption and core performance:

- Arm® Cortex®-M7 (CPU1) located in the D1 domain and operating at up to 480 MHz
- Arm® Cortex®-M4 (CPU2) located in the D2 domain and operating at up to 240 MHz

The purpose of this application note is to highlight STM32H747/757 performance and explain how to make the most of their flexible architecture to reduce power consumption. The document is split into four parts:

1. Introduction to the STM32H747/757 power management concepts and voltage regulators, peripheral allocation by both CPUs, as well as low-power mode entry and exit.
2. Illustration of how to use the above features to reduce power consumption through a temperature acquisition use case based on the STM32H747I-DISCO Discovery kit and the X-NUCLEO-IKS01A2 expansion board with three scenarios:
   – D1 domain in DRun mode, D2 domain in DStop and D3 domain in Run mode.
   – D1 domain in DStop mode, D2 domain in DStop and D3 domain in Autonomous mode.
   – D1 domain in DStop domain, D2 domain in DStop and D3 domain Run/Stop mode.
3. Graphical display of temperature values using the StemWin library and FreeRTOS™.
4. Cortex®-M7 and Cortex®-M4 core synchronization using a semaphore.

Refer to the following documents for further information on the STM32H747/757 line:

- *STM32H745/755 and STM32H747/757 advanced Arm®-based 32-bit MCUs* reference manual (RM0399).
- STM32H747xx and STM32H757xx datasheets.
- STM32H747I-DISCO Discovery kit user manual: *Discovery kit with STM32H747XI MCU* (UM2411).

# 1 General information

This document applies to STM32H747/757 Arm®-based devices.

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

# 2 System architecture

## 2.1 System architecture overview

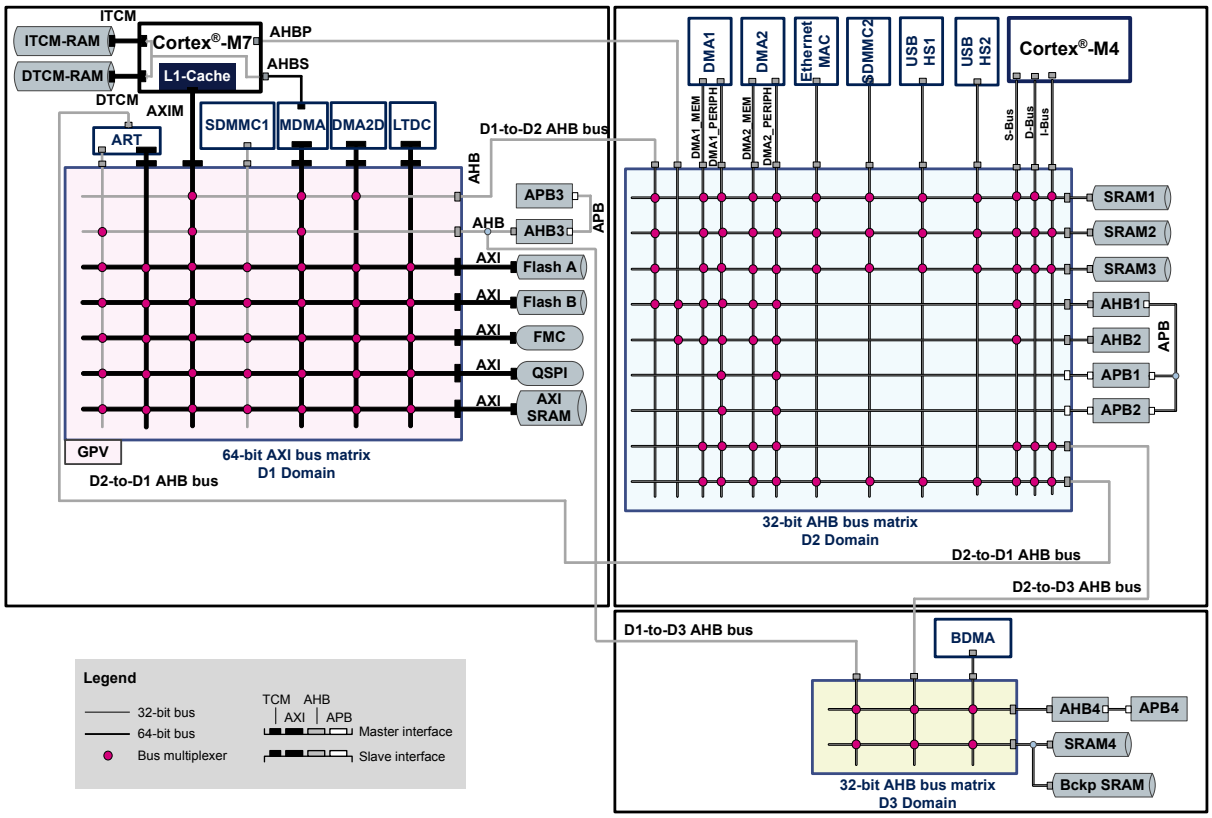
The STM32H747/757 line embeds a flexible power architecture dedicated to power consumption reduction. The devices embed two power regulators: an LDO and a high power-efficiency SMPS step-down converter. Each regulator can be selected as power source for the product power domains.

Three power domains can be independently switched ON and OFF, depending on application needs:

- **D1 domain**: this domain contains the Cortex®-M7 core, the Flash memory and some peripherals. Thanks to the AXI bus matrix, this domain encompasses high bandwidth features and smart management.
- **D2 domain**: this domain contains the Cortex®-M4 core, most of the memories dedicated to I/O processing and most of the peripherals that are less bandwidth demanding.
- **D3 domain**: this domain contains the system control, low-power peripherals and memories designed to manage low-power modes. It is designed to be autonomous, and embeds a 64-Kbyte RAM, a basic DMA controller (BDMA), plus low-power peripherals to run basic functions while D1 and D2 domains can be switched off to save power.

Figure 1 illustrates STM32H747/757 system architecture.

**Figure 1. STM32H747/757 system architecture**



## 2.2 System supply configuration

D1, D2 and system D3 domains are supplied at once from a single regulator. The selected regulator provides a common $V_{CORE}$ level depending on the system operating mode (**Run**, **Stop**, and **Standby**).

The VCORE domain supply can be provided by the **SMPS step-down converter**, **voltage regulator** or by an **external supply** (VCAP). VCORE supplies all the digital circuitries except for the Backup domain and the Standby circuitry.

The different supply configurations are controlled by the **LDOEN**, **SDEN**, **SDEXTHP**, **SDLEVEL** and **BYPASS** bits in PWR control register 3 (**PWR_CR3**).

### 2.2.1 Voltage regulator (LDO)

The voltage regulator (LDO) can provide three different operating modes, Main (MR), Low-power (LP) or Off mode, depending on the application needs. The LDO adjusts the output voltage through six power supply levels. The table below summarizes the regulator LDO operating modes.

**Table 1. LDO converter operating modes and voltage regulators**

| Operating mode | Regulator operating mode | Voltage scaling | Output voltage (V) | Maximum output frequency | Description |
|---|---|---|---|---|---|
| Run | MR | VOS0 | 1.35 | 480 MHz | Boosted performance |
| | | VOS1 | 1.2 | 400 MHz | High performance |
| | | VOS2 | 1.1 | 300 MHz | Medium performance and consumption |
| | | VOS3(1) | 1.0 | 200 MHz | Optimized performance and low-power consumption. |
| Stop | MR or LP | SVOS3 | 1.0 | NA | The regulator can be:<br>• In **Main** mode to allow fast exit from Stop mode.<br>• In **LP** mode to obtain a lower $V_{CORE}$ supply level and extend the exit-from-Stop latency. |
| | LP | SVOS4 | 0.9 | | More power consumption.<br>In SVOS4/SVOS5, the contents of the registers and memories are retained. |
| | | SVOS5 | 0.7 | | More power consumption.<br>In SVOS4/SVOS5, the contents of the registers and memories are retained. |
| Standby | OFF | NA | NA | NA | The regulator is off.<br>The contents of the registers and memories are lost except for the **Standby circuitry** and the **Backup domain**. |

1. *By default, VOS3 is selected after system reset.*

### 2.2.2 SMPS step-down converter

The SMPS step-down converter is a high power-efficient SMPS step-down converter and non-linear switching regulator that allows to achieve a lower power consumption compared to a conventional voltage regulator. It can be used in internal or external supply mode (VDD_extern):

- The internal supply mode is used to directly supply the VCORE domain. The SMPS step-down converter behaves in the same way as the LDO regulator.
- The external supply mode can generate an intermediate supply level (VDD_extern at 1.8 or 2.5 V), which can supply the voltage regulator (LDO) and optionally an external circuitry. When the SMPS step-down converter supplies an external circuitry, it is forced ON and operates in MR mode. The intermediate voltage level is selected through **SDLEVEL** bits in PWR control register 3 (**PWR_CR3**).

## 2.3 Peripheral allocation

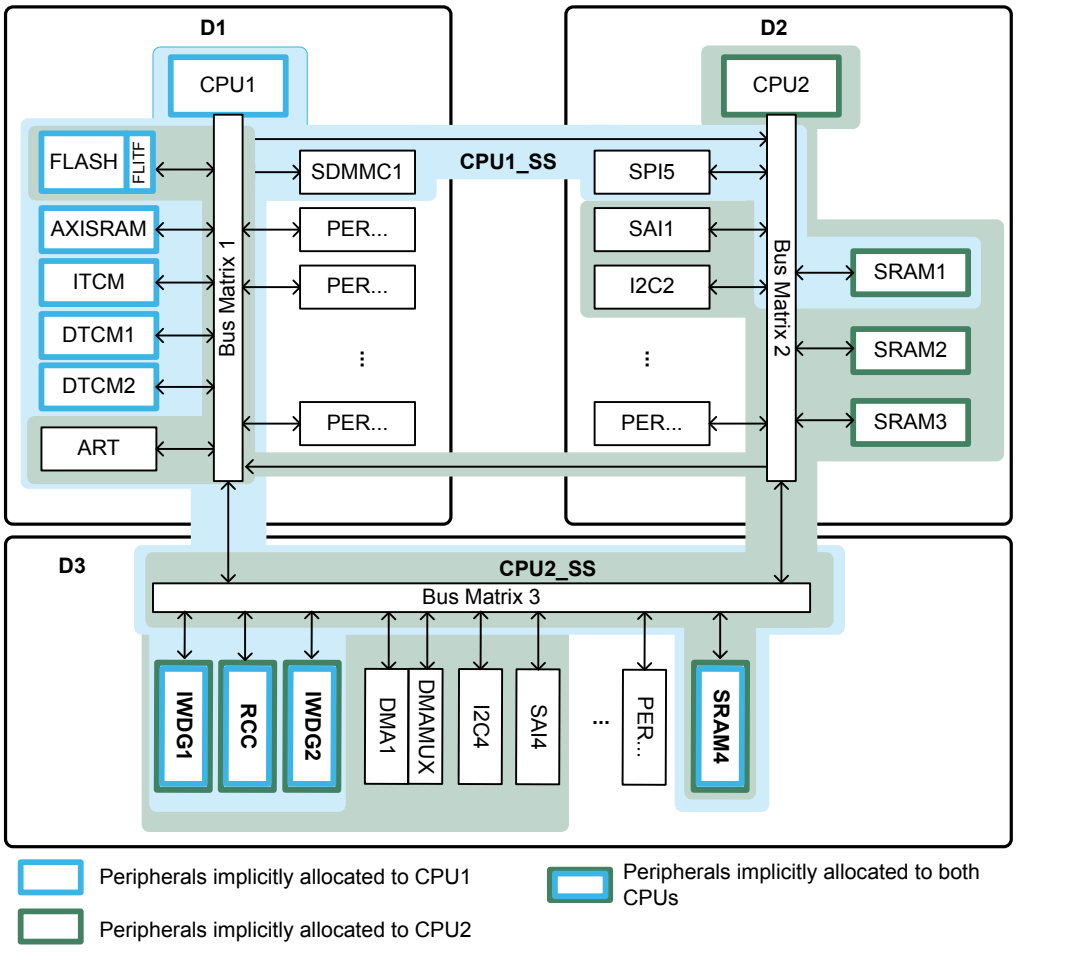
### 2.3.1 Peripheral allocation overview

The peripheral allocation is used by the reset and clock controller (RCC), to automatically control the clock gating according to the CPUs and domain modes, and by the power controller (PWR) to control the supply voltages of D1, D2 and D3 domains.

Each CPU can allocate peripherals for itself (or optionally for the other CPU). To do this, each CPU features dedicated registers and bits in order to perform peripheral allocation: the PERxEN bits of RCC_C1_xxxxENR and RCC_C2_xxxxENR. These bits are also used to link peripherals to a CPU and build a CPU peripheral subsystem so that each CPU can control its peripheral subsystem kernel and bus interface clock.

The CPU peripheral subsystem clocks follow the CPU state: for example, they are clocked when the CPU is in CRun mode or clock-gated when the CPU is in CStop mode.

**Figure 2. Peripheral allocation**




Some peripherals are allocated by default to the CPUs:

- FLASH, AXISRAM, ITCM, DTCM1, DTCM2 and SRAM4 are allocated to CPU1.
- SRAM1, SRAM2, SRAM3 and SRAM4 are allocated to CPU2.
- SRAM4, IWGD1, IWGD2 and RCC are common resources and are allocated both to CPU1 and CPU2.

### 2.3.2 Allocating peripherals belonging to D1 and D2 domains

The kernel clock is provided to the peripherals located in **D1** and **D2** domains when one of the following conditions is met:

1. The **CPU** to which the peripheral is allocated is in **CRun** mode.

2. The **CPU** to which the peripheral is allocated is in CSleep mode and **PERxLPEN** is set to 1.
3. The **CPU** to which the peripheral is allocated is in **CStop** mode with **PERxLPEN** set to 1, the peripheral generates a kernel clock request, and the selected clock is **hsi_ker_ck** or **csi_ker_ck**.
4. The **CPU** to which the peripheral is allocated is in **CStop** mode with **PERxLPEN** set to 1, and the peripheral kernel source clock is **lse_ck** or **lsi_ck**.

The bus interface clock is provided to the peripherals only when conditions 1 and 2 are met.

### 2.3.3 Allocating peripherals belonging to D3 domain

The D3 Autonomous mode allows the delivery of the peripheral clocks to peripherals located in the D3 domain, even if the CPU to which they are allocated is in CStop mode. When a peripheral has its autonomous bit enabled, it receives its peripheral clock according to D3 domain state independently from the state of the CPU to which it is allocated (for instance CPU in CStop mode):

- If the D3 domain is in DRun mode, the peripherals that are in Autonomous mode receive their peripheral clock.
- If the D3 domain is in DStop mode, no peripheral clock is provided.

The kernel clock is provided to the peripherals located in the D3 domain if the following conditions are met:

1. The CPU to which the peripheral is allocated is in CRun mode.
2. The CPU to which the peripheral is allocated is in CSleep mode and PERxLPEN is set to 1.
3. The CPU to which the peripheral is allocated is in CStop mode and the D3 domain is in DRun mode with PERxAMEN set to 1.
4. The CPU to which the peripheral is allocated is in CStop mode, the D3 domain is in DStop mode with PERxAMEN set to 1, the peripheral generates a kernel clock request, and the kernel clock source is hsi_ker_ck or csi_ker_ck.
5. The CPU to which the peripheral is allocated is in CStop mode, D3 domain is in DStop mode with PERxAMEN set to 1, and the kernel clock source of the peripheral is lse_ck or lsi_ck.

The bus interface clock is provided to the peripherals only when condition 1, 2 or 3 is met.

## 2.4 Operating modes

### 2.4.1 Operating modes

The operating modes allow controlling the clock distribution to the different system blocks and powering them. Several operating modes are available when using STM32H747/757 device line. It is up to the user to choose the best compromise between low-power consumption, short start-up time and available wake-up sources. The system operating mode is driven by CPU1 subsystem, CPU2 subsystem and system D3 autonomous wake-up.

**Table 2. Operating mode overview**

| CPU subsystems and domains | Modes | Description |
|---|---|---|
| CPU subsystems | CRun | The CPU and CPU subsystem peripheral allocated via **RCC PERxEN** bits are clocked. |
| | CSleep | The CPU clocks are stalled and the CPU subsystem allocated peripheral clock(s) operate according to **RCC PERxLPEN**. |
| | CStop | The CPU and CPU subsystem peripheral clocks are stalled. |
| D1 and D2 subsystems | DRun | The domain bus matrix is clocked:<br>• The domain CPU subsystem is in CRun or CSleep mode,<br>or<br>• the other domain CPU subsystem having an allocated peripheral in the domain is in CRun or CSleep mode. |
| | DStop | The domain bus matrix clock is stalled:<br>• The domain CPU subsystem is in CStop mode,<br>and |

| CPU subsystems and domains | Modes | Description |
|---|---|---|
| D1 and D2 subsystems | | • the other domain CPU subsystem has no peripheral allocated in the domain or the other domain CPU subsystem having an allocated peripheral in the domain is also in CStop mode,<br>and<br>• at least one PDDS_Dn bit for the domain selects DStop mode. |
| | DStandby | The domain is powered down:<br>• The domain CPU subsystem is in CStop mode,<br>and<br>• the other domain CPU subsystem has no peripheral allocated in the domain or the other domain CPU subsystem having an allocated peripheral in the domain is also in CStop mode,<br>and<br>• all PDDS_Dn bits for the domain select DStandby mode. |
| System / D3 subsystems | Run | The system clock and D3 domain bus matrix clock are running:<br>• A CPU subsystem is in CRun or CSleep mode<br>• or a wake-up signal is active. (i.e. System D3 autonomous mode) |
| | Stop | The system clock and D3 domain bus matrix clock are stalled:<br>• Both CPU subsystems are in CStop mode,<br>and<br>• all wake-up signals are inactive.<br>and<br>• at least one PDDS_Dn bit for any domain selects Stop mode. |
| | Standby | The system is powered down:<br>• Both CPU subsystems are in CStop mode,<br>and<br>• all wake-up signals are inactive,<br>and<br>• all PDDS_Dn bits for all domains select Standby mode. |

### 2.4.2 Entering/exiting low-power modes

Depending on the application requirements, several low-power modes can be selected to reduce power consumption. The table below summarizes the low-power entry and exit conditions:

**Table 3. Summary of low-power mode entry/exit conditions**

| Low-power mode | Mode entry | CPU and CPU subsystem peripheral allocated via |
|---|---|---|
| **CSleep** | WFI or return from ISR | Any Interrupt enabled in NVIC |
| | WFE was used for entry and SEVONPEND = 0 | Any event. |
| | WFE was used for entry and SEVONPEND = 1 | Any interrupt even when disabled in NVIC or any event. |
| **CStop** | WFI or return from ISR | EXTI Interrupt enabled in NVIC. |
| | WFE was used for entry and SEVONPEND = 0 | EXTI event. |
| | WFE was used for entry and SEVONPEND = 1 | • EXTI Interrupt even when disabled in NVIC,<br>• or EXTI event. |
| **DStop** | • The domain CPU subsystem enters CStop and the other domain CPU subsystem has no peripheral allocated in the domain, or it is also in CStop mode.<br>• The other domain CPU subsystem has an allocated peripheral and enters CStop and the domain CPU subsystem is in CStop mode.<br>• The other domain CPU subsystem deallocated its last peripheral in the domain and the domain CPU subsystem is in CStop mode.<br>• At least one PDDS_Dn bit for the domain selects Stop mode. | • The domain CPU subsystem exits CStop mode.<br>• The other domain CPU subsystem has an allocated peripheral in the domain and exits CStop mode.<br>• The other domain CPU subsystem allocates a first peripheral in the domain. |
| **Stop** | • CPU1 and CPU2 are in CStop mode.<br>• There is no active EXTI wake-up source and Run_D3 = 0.<br>• At least one PDDS_Dn bit for any domain selects Stop mode. | An EXTI wake-up. |
| **DStandby** | • The domain CPU subsystem enters CStop mode and the other domain CPU subsystem has no peripheral allocated in the domain or is also in CStop mode.<br>and<br>• The other domain CPU subsystem has an allocated peripheral and enters CStop mode, and the domain CPU subsystem is in CStop mode.<br>• The other domain CPU subsystem deallocated its last peripheral in the domain and the domain CPU subsystem is in CStop mode.<br>• All PDDS_Dn bits for the domain select Standby mode.<br>• All WKUPF bits in Power Control/Status register (PWR_WKUPFR) are cleared. | • The domain CPU subsystem exits CStop mode.<br>• The other domain CPU subsystem has an allocated peripheral in the domain and exits CStop mode.<br>• The other domain CPU subsystem allocates a first peripheral in the domain. |
| **Standby** | • The CPU1 and CPU2 subsystems are in CStop mode, there is no active EXTI wake-up source, and RUN_D3 = 0.<br>• All PDDS_Dn bits for all domains select Standby mode.<br>• All WKUPF bits in Power Control/Status register (PWR_WKUPFR) are cleared. | WKUP pins rising or falling edge.<br>RTC alarm (Alarm A and Alarm B) or RTC wake-up.<br>TAMPER event, TIMESTAMP event.<br>External reset in NRST pin, IWDG reset. |

# 3 Application use case

The application use case described in this section illustrates the low-power temperature acquisition using the STM32H747I-DISCO Discovery kit and the X-NUCLEO-IKS01A2 expansion board. It provides guidelines on how to implement a low-power application taking benefits from STM32H747/757 power domains.

In this basic use case, the temperature acquisition is handled by the Arm® Cortex®-M4 core, while the display of temperature values is controlled by the Cortex®-M7 core. The two cores are synchronized using hardware semaphores.

## 3.1 Arm® Cortex®-M4 tasks

### 3.1.1 Low-power temperature acquisition

#### *3.1.1.1 Acquisition principles*

This application note provides a temperature acquisition use case based on the Cortex®-M4 and D3 domain peripherals, with different domain power states. In this use case, the X-NUCLEO-IKS01A2 expansion board is used to acquire temperature values from a HTS221 sensor via the $I^2C$ bus. Three different modes are proposed in order to optimize power consumption and highlight STM32H747/757 advanced power management:

- **Mode 1**
  The Cortex®-M4 core first allocates Flash memory to boot the execution code from it. This prevents the D1 domain to enter DStop mode if the Arm® Cortex®-M7 enters CStop. Dedicated peripherals located in the D3 domain are used during the acquisition process. As a result, only the D2 domain enters DStop when the Arm® Cortex®-M4 enters CStop.
- **Mode 2**
  To achieve additional power consumption reduction, the Flash memory is deallocated for the Arm® Cortex®-M4 core to enable D1 domain entry to DStop mode independently from CPU2 state. The Arm® Cortex®-M4 core first boots from Flash. The acquisition code is then executed from an SRAM area located in the D2 domain. This mode illustrates the D3 Autonomous mode which allows D3 domain peripherals to run independently in Autonomous mode even when both CPUs are in CStop mode.
- **Mode 3**
  Further power consumption saving can be achieved by switching D3 domain between DRun and DStop modes during data transfer.

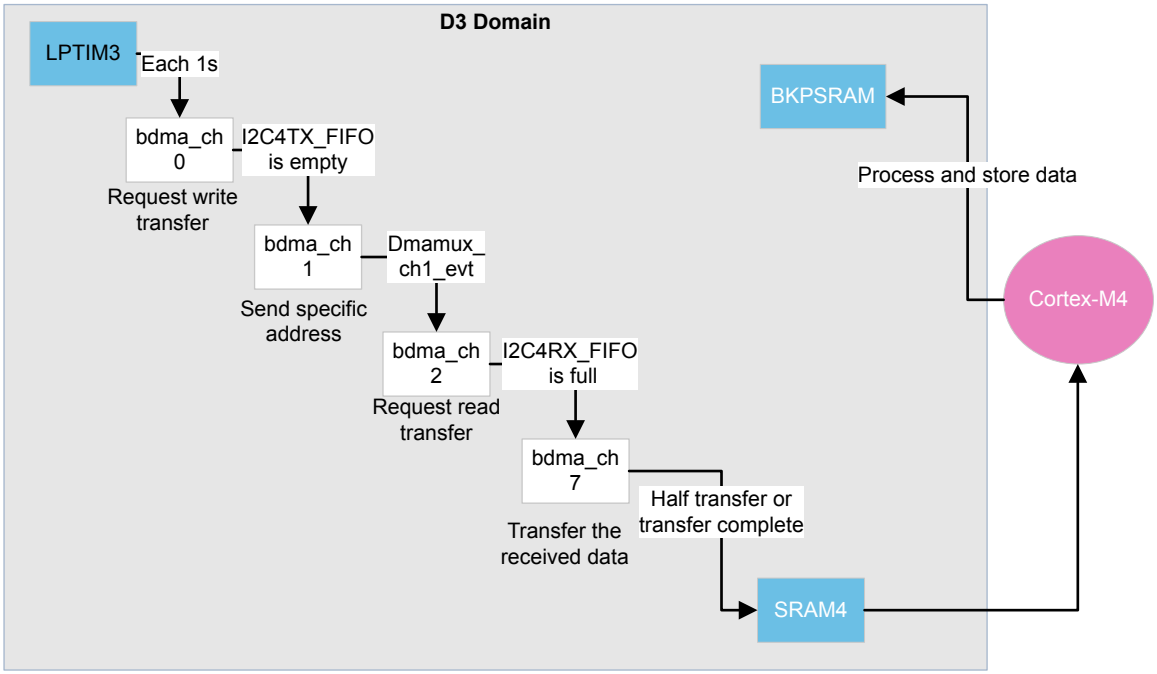
#### *3.1.1.2 Description of the transmission process*

Below a detailed description of the transmission process:

1. The rising edge of lptim3_out signal triggers BDMA channel 0 (bdma_ch0) once per second to begin data transfer in order to configure the I2C4 interface to request write transfer to the HTS221 sensor.
2. As soon as I2C4 TX-FIFO is empty, BDMA channel 1 (bdma_ch1) begins data transfer from SRAM4 to I2C4_TXDR register to send a specified address to the HTS221 sensor.
3. The end of this transfer triggers BDMA channel 2 data (bdma_ch2) transfer from SRAM4 to I2C4_CR2 register to request read transfers from the sensor.
4. BDMA channel 7 (bdma_ch7) is then used to transfer the received data from I2C4_RXDR register (data sent by the HTS221 sensor) to SRAM4. bdma_ch7 data transmission is enabled by setting RXDMAEN bit in I2C_CR1 register. bdma_ch7 is configured to generate interrupts to wake up the CPU from CStop mode each time a half-transfer is complete (10 bytes of data) or when a full transfer is complete (20 bytes of data).

Refer to Figure 3 for an overview of the transmission process:

Figure 3. **Transmission process**




### 3.1.1.3 *Acquisition mode implementation*

Three different acquisition modes are possible: mode 1, mode 2 and mode 3.

**Mode 1**

In mode 1, the Flash memory (located in D1 domain) is allocated to CPU2 (Arm® Cortex®-M4). The D1 domain consequently takes into account CPU2 operating modes, for example by keeping D1 domain in DRun mode when CPU2 is in CRun. As a result, the Flash memory can be deallocated after making sure that code booting by CPU2 is complete.

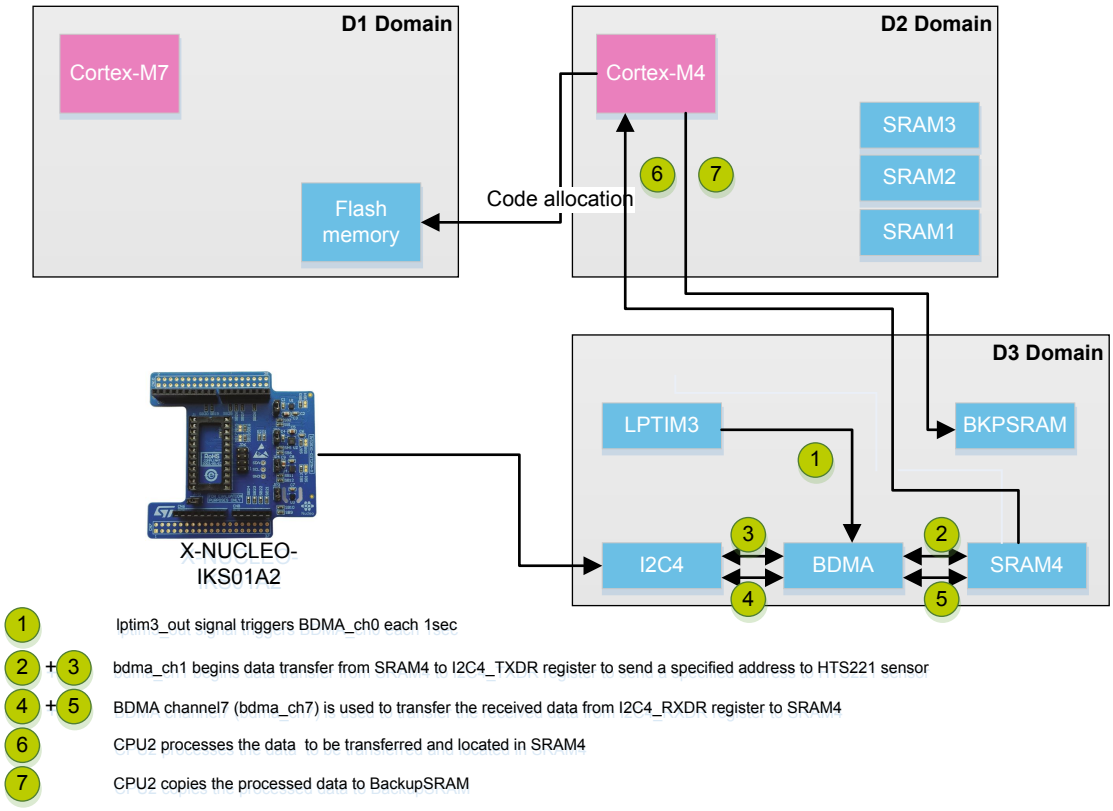
Below the main temperature acquisition steps in mode 1:

1. Arm® Cortex®-M4 booting from Flash memory.
2. Temperature acquisition from shield to SRAM4 through I2C4 located in the D3 domain.
3. Data transfer from SRAM4 to Backup SRAM after waking up CPU2.

Table 4. **Domain operating modes in mode 1**

| Power domain | Mode |
|---|---|
| D1 domain | DRun |
| D2 domain | DStop |
| D3 domain | Run |

Figure 4. **Acquisition process in mode 1**




### Mode 2

In mode 2, the CPU2 executes code from SRAM1 located in the D2 domain, so that D1 domain can enter DStop mode independently from CPU2 operating mode.

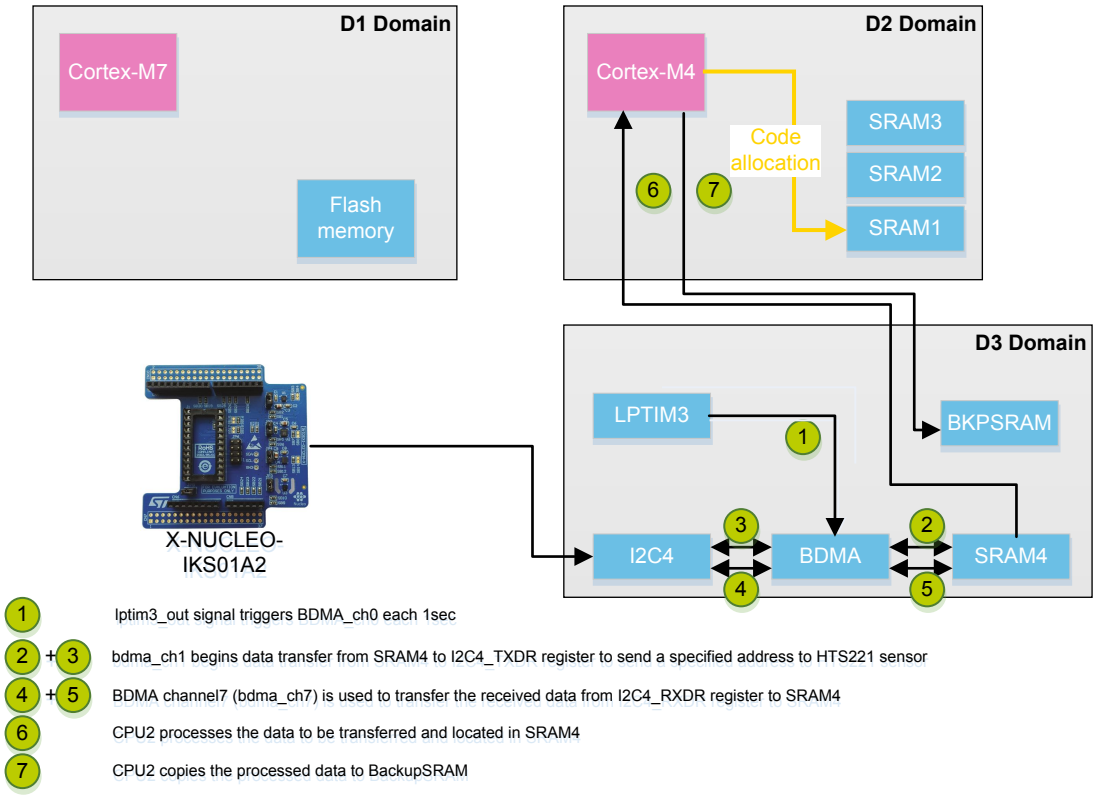
Below the main temperature acquisition steps in mode 2:

1. CPU2 executing from SRAM1 located in the D2 domain.
2. Temperature acquisition from shield to SRAM4 through I2C4 located in the D3 domain.
3. Data transfer from SRAM4 to Backup SRAM after waking up CPU2.

Table 5. **Domain operating modes in mode 2**

| Power domain | Mode |
|---|---|
| D1 domain | DStop |
| D2 domain | DStop |
| D3 domain | Run |

Figure 5. **Acquisition process in mode 2**




### Mode 3

The DMAMUX2 event 7 signal (dmamux2_evt7) is used by the EXTI to clear the pending request and switch back the D3 domain to Stop mode every time a half transfer or a transfer complete of data is complete.

For complementary information on the acquisition process, please refer to application note *STM32H7x3 smart power management software expansion for STM32Cube* (AN5014) available from www.st.com:

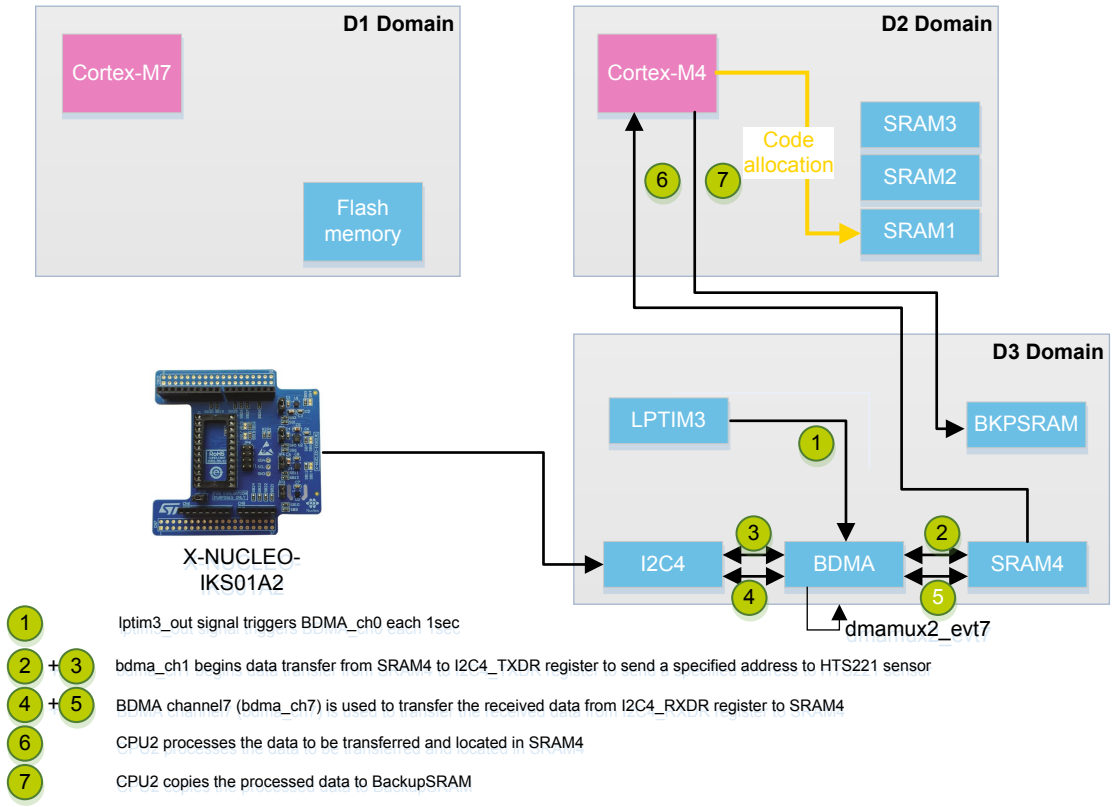
Below the main temperature acquisition steps in mode 3:

1. CPU2 executing from SRAM1 located in the D2 domain.
2. Temperature acquisition from shield to SRAM4 through I2C4 located in the D3 domain.
3. Data transfer from SRAM4 to Backup SRAM after waking up CPU2.

Table 6. **Domain operating modes in mode 3**

| Power domain | Mode |
|---|---|
| D1 domain | DStop |
| D2 domain | DStop |
| D3 domain | Run/Stop |

Figure 6. **Acquisition process in mode 3**



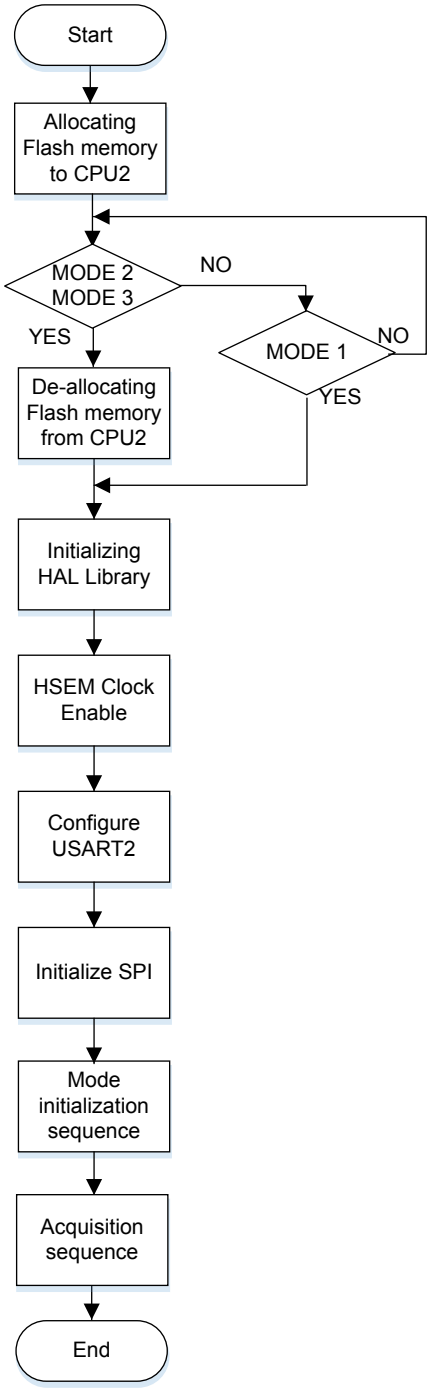

### 3.1.2 Detailed description of temperature acquisition in different modes

#### 3.1.2.1 *Temperature acquisition in different modes*

The sequence below must be executed by the Arm® Cortex®-M4 to perform temperature acquisition (see Figure 7 for an overview of the temperature acquisition sequence depending on the mode):

1. To boot the application code from Flash memory, allocate the Flash memory (located in D1 domain) in the *system_stm32h7xx.c* file. The D1 domain consequently takes into account the CPU2 or CPU2 subsystem operating modes. Since the D1 domain enters DStop mode in mode 2 and 3, the Flash memory must be deallocated for these two modes, and CPU2 proceeds with code execution from SRAM1 (located in D2 domain).
2. Initialize the hardware abstraction layer (HAL) library.
3. Enable hardware semaphore (HSEM).
4. Configure USART2 to display message on Hercules terminal.
5. Execute the initialization and acquisition tasks described in detailed in Figure 8 and Figure 9.

**Figure 7. Global flowchart of temperature acquisition depending on modes**



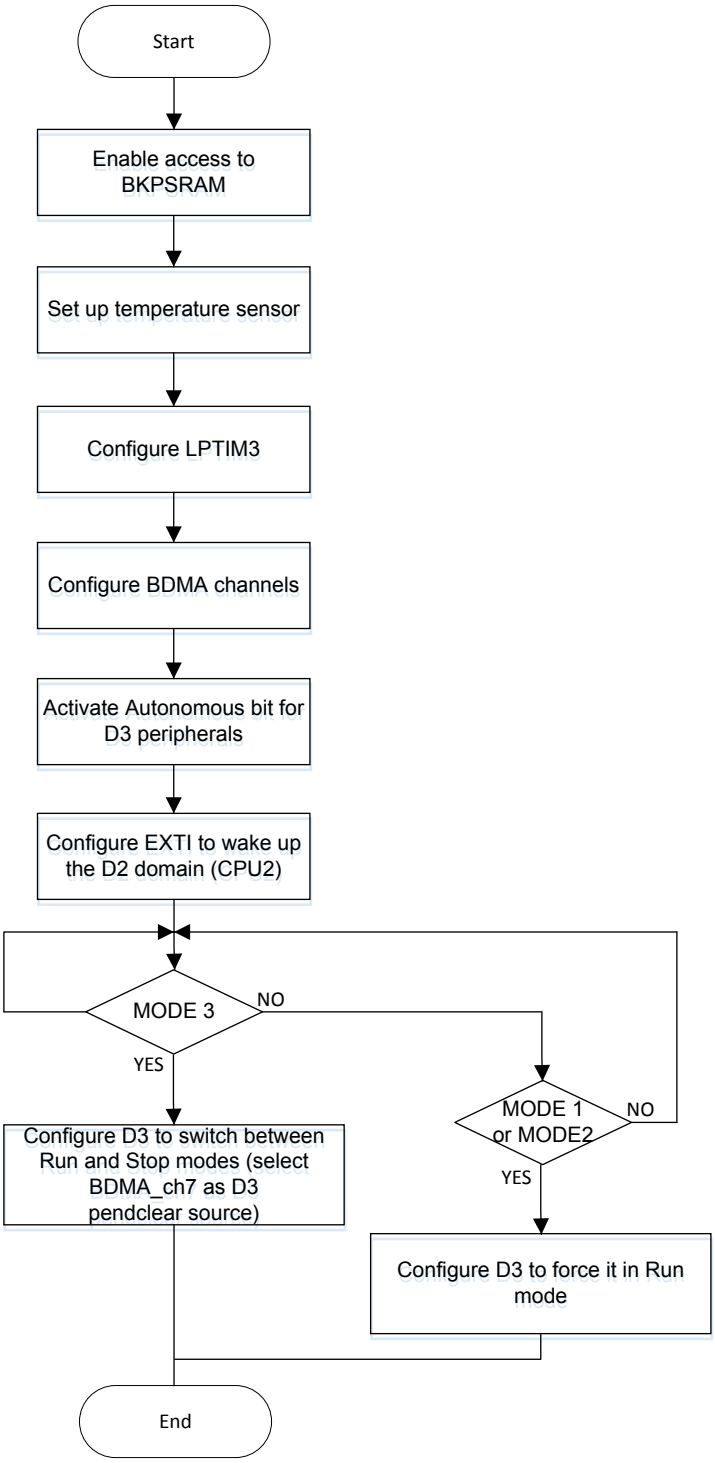
### Mode initialization sequence

To initialize the acquisition process, accesses to BKPSRAM must be enabled as well as the temperature sensor. The used peripherals must then be configured and the corresponding autonomous bits set in RCC_D3AMPR register. The BDMA channel 7 interrupt (bdma_ch7_it) is used to wake up the D2 domain from DStop mode.

In mode 3, since D3 domain toggles between Run and Stop modes, bdma_ch7_it event is selected as D3 domain pendclear source contrary to mode 1 and mode 2 where the application has to keep the D3 domain in Run mode.

Refer to Figure 8 for a detailed description of the initialization sequence.

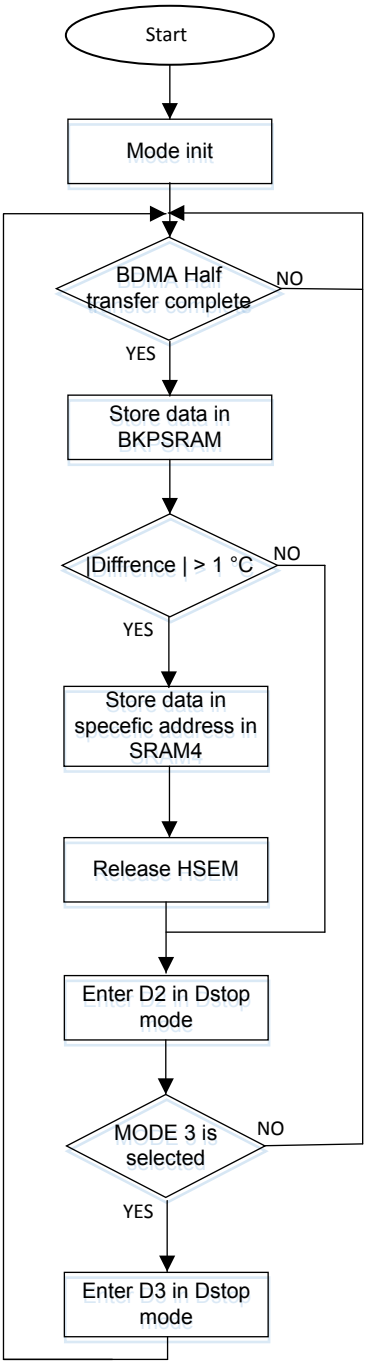
Figure 8. **Mode initialization sequence**




### Acquisition sequence

Figure 9 provides a detailed description of the acquisition sequence and highlights the fact that the D3 domain must also enter Stop mode in mode 3.

**Figure 9. Acquisition sequence**



## 3.2 Arm® Cortex®-M7 tasks

### 3.2.1 DSI Host interface

STM32H7x7 microcontrollers embed a DSI Host interface, a high-speed communication channel that allows the microcontroller to communicate with a graphical display with a reduced pincount and a very high communication speed up to 1 Gbits/s (up to 500Mbits/s per lane). The DSI Host interface can be configured in Video mode or in Adapted command mode.

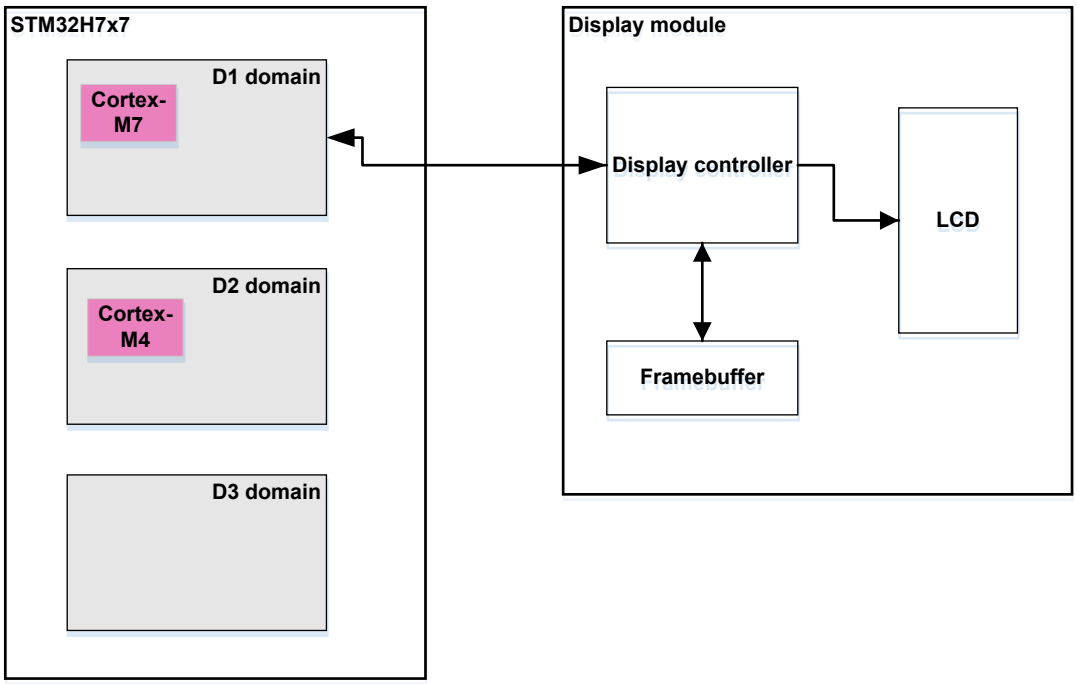
To allow CPU1 to enter low-power mode while going on refreshing the display normally, the Adapted command mode has been selected in the application use case.

Refer to Table 7 for a summary of the difference between the two modes.

**Table 7. Embedded graphic system**

| Display mode | Description |
|---|---|
| **Video mode** | The MCU sends a real-time pixel stream of data plus video timing information in order to refresh the display. |
| **Adapted command mode** | The display controller handles the refresh operation. The display relies on its internal controller and framebuffer to perform the refresh operation without MCU intervention. |

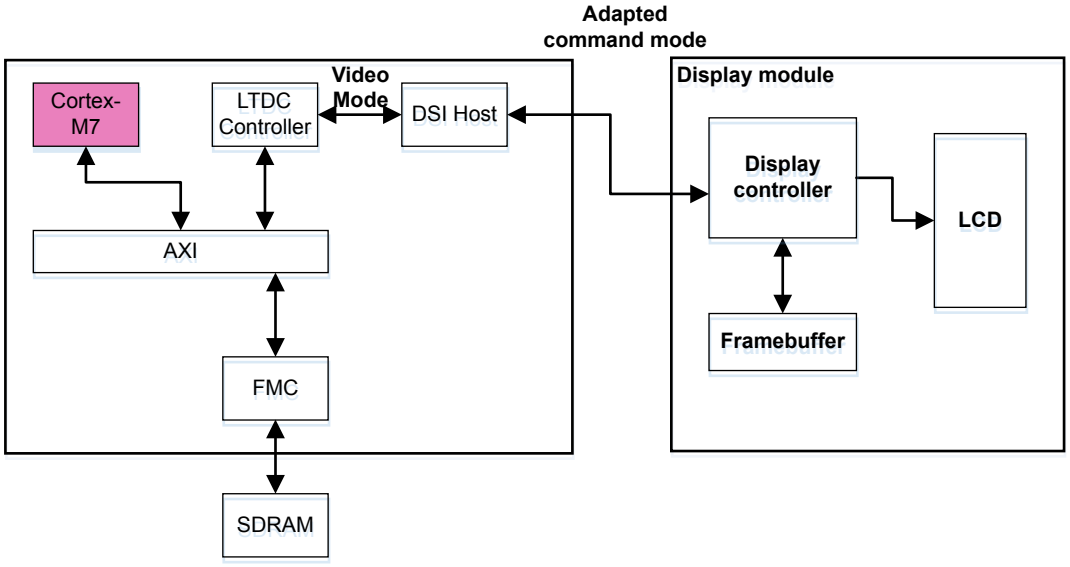
**Figure 10. High-level block diagram of STM32H7x7 and DSI interface**




### 3.2.2 Detailed description of the displaying process

In the application use case described in this document, the LCD-TFT display controller (LTDC) fetches data from SDRAM bank 2 via the FMC controller and sends them to the DSI controller in Video mode. As soon as the DSI Host has captured one full frame coming from the LTDC, it transforms it into a series of write commands to update the display GRAM, and then disables the LTDC. The DSI Host controller automatically refreshes the GRAM with the LTDC without CPU intervention. Figure 11 shows the displaying sequence.

Figure 11. **Displaying sequence in Adapted command mode**




### 3.2.3 STemWin Library

The STemWin library is a professional graphical stack library enabling the building up of graphical user interfaces (GUIs) with any STM32 microcontroller, LCD-TFT display and LCD-TFT controller, taking advantage of STM32 hardware accelerations whenever possible.

#### *3.2.3.1 STemWin configuration*

STemWin configuration is divided into two parts:

- **GUI configuration:** configuration of default colors and fonts of the available memory
- **LTDC configuration**: definition of hardware-dependent graphical parameters, display driver and color conversion routines to be used

When a new LTDC controller needs to be supported, two essential files must be created/modified: GUIConf.c and LCDConf.c:

- ***GUIConf.c***
  In this file, the user must implement the `GUI_X_Config()` function which is the very first routine called during the initialization process. Its main task is to set up the available memory for the GUI, and then assign it to the dynamic memory management system. This operation is performed via the `GUI_ALLOC_AssignMemory()` function. It passes a pointer to a memory block and its size (in bytes) to the memory manager:
- ***LCDConf.c***
  In this file, the main function is `LCD_X_Config()`, called immediately after `GUI_X_Config()` has been executed. `LCD_X_Config()` creates and configures a display driver for each layer by calling:
  - `GUI_DEVICE_CreateAndLink()`: to create the driver device and links it to the device chain
  - `LCD_SetSizeEx()` and `LCD_SetVSizeEx()`: to configure the display size
  - `LCD_SetVRAMAddrEx()`, required for linear addressable memory.

#### *3.2.3.2 StemWin APIs*

StemWin library includes APIs that can be used to display strings and values, or program parameters.

**Displaying strings and values**

To display values, the application can use standard C library strings and functions, such as `sprintf`, or call a routine that displays the strings and values mentioned in table below:

**Table 8. Displaying Strings and values APIs**

| Routine | Description |
|---|---|
| `GUI_DispFloat()` | Displays a floating point value with a specified number of characters at the current text position in the current window using the current font. |
| `GUI_DispStringAt()` | Displays the string passed as parameter at a specified position in the current window using the current font. |

#### Writing parameters

To get a better visibility for the application, the application can control the way of writing by choosing the text font, the color and the background color. The following table shows the routine that can be used by the application

**Table 9. Routines used to set parameters**

| Routine | Description |
|---|---|
| `GUI_SetColor` | Sets the current foreground color. |
| `GUI_SetFont` | Sets the font to be used to output text. |
| `GUI_SetBkColor` | Sets the default background color. |
| `GUI_SetLayerVisEx` | Sets the visibility of the given layer. |

### 3.2.4 JPEG image decoding

To display an overview image on the screen, a JPEG image stored in the internal Flash memory is decoded using the JPEG hardware decoder in DMA mode. The application can create the JPEG image using any drawing software, convert it into a source file using the Arm file converter with µVision®, and add it to the project. To properly decode a JPEG stream the application must configure the following functions:

- **`JPEG_Decode_DMA`**
  This function starts JPEG decoding in DMA mode. It is split into two subfunctions:
  - Initialization of the decoding process:
    This function configures the JPEG Codec in decoding mode, stops JPEG processing, disables all interrupts, flushes input and output FIFOs, and finally enables end-of-conversation and end-of-header parsing interrupts.
  - JPEG decoding using DMA:
    This function configures the JPEG MDMA input/output transfer callbacks, MDMA transfer size (it must be a multiple of MDMA buffer size) and finally starts MDMA FIFO input/output transfers to the YCbCr destination frame buffer address in Interrupt mode.
  - `HAL_JPEG_GetInfo`
    This function extracts the image configuration from the JPEG header at the end of JPEG decoding. It reads the configuration parameters (color space, image height, image width and image quality).
- **`DMA2D_CopyBuffer`**
  This function is used to copy decoded RGB data to the display framebuffer.
  It first configures DMA2D transfer mode as memory-to-memory with pixel format conversion, sets RGB565 as DMA2D color mode, sets DMA2D_INPUT_YCBCR as foreground input color mode, and then selects chroma sub-sampling 4:2:2 for half resolution horizontally on the chroma components.
  Finally, this function copies the new decoded frame to the LTDC framebuffer and start the DMA2D transfer.

### 3.2.5 Arm® Cortex®-M7 programming

The main purposes of the graphical application executed by the Arm® Cortex®-M7 are:

- to display an image decoded with JPEG decoder in DMA mode.
- to configure the push-button to be able to interrupt CPU1 core.
- to start displaying temperature values once the push-button is pressed, .

Figure 12 describes all the APIs that are used in the application program and Table 10 shows the global graphical application flowchart.

Figure 12. **Arm® Cortex®-M7 programming flowchart**

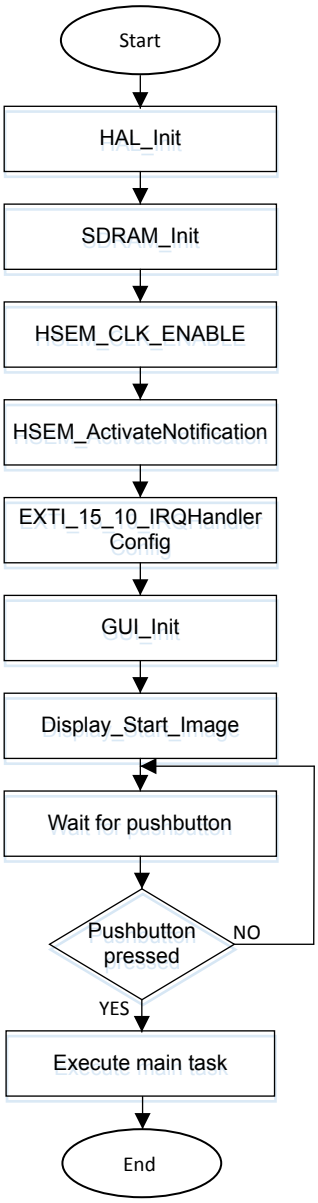



Table 10. **Description of the APIs used by Arm® Cortex®-M7 core**

| API | Description |
|---|---|
| `HAL_Init` | Initialize the STM32H7xx HAL Library |
| `BSP_SDRAM_Init` | Initializes the SDRAM device |
| `__HAL_RCC_HSEM_CLK_ENABLE` | Enables hardware semaphores |
| `HAL_HSEM_ActivateNotification` | Activates HSEM notification to the Arm® Cortex®-M4 core |
| `HAL_JPEG_Init` | Initializes JPEG decoding in DMA mode |
| `EXTI15_10_IRQHandler_Config` | Configures the push-button to interrupt CPU1 (Arm® Cortex®-M7). |
| `GUI_Init()` | Initializes the STemWin GUI library |

| API | Description |
|---|---|
| `JPEG_Decode_DMA` | Starts JPEG decoding with DMA processing |
| `HAL_JPEG_GetInfo` | Extracts the image configuration from the JPEG header during the decoding process |
| `DMA2D_CopyBuffer` | Copies RGB decoded Data to the display Frame buffer |

### 3.2.6 Main Arm® Cortex®-M7 tasks using FreeRTOS™

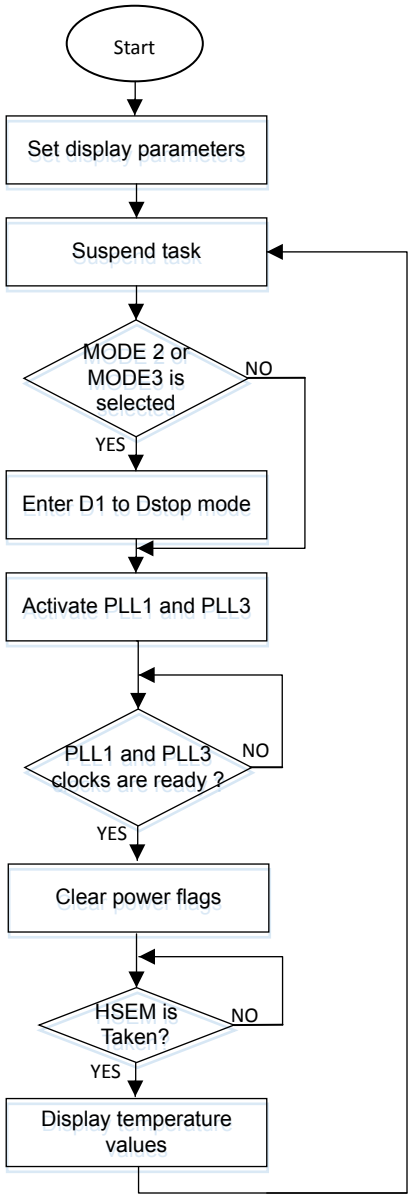
The Arm® Cortex®-M7 core first configures the display settings by selecting the current layer and its visibility, and by choosing fonts as well as the background and foreground colors.

The D1 domain then enters DStop mode if the selected mode is mode 2 or mode 3, and the system enters Stop mode to reduce power consumption.

Since PLLs are cleared by hardware when entering low-power mode, the program must reactivate them and wait until their clock state become ready. When the Arm® Cortex®-M4 core releases the semaphore, its wakes up the Arm® Cortex®-M7 who takes the semaphore and displays the temperature value which is stored in shared memory (SRAM4 at address 0x3800 8004).

Figure 13 shows Arm® Cortex®-M7 main tasks.

Figure 13. **Arm® Cortex®-M7 main task flowchart**




## 3.3 Core synchronization using hardware semaphore

In a parallel programming environment, a semaphore is a synchronization object that controls the access to a common resource by multiple processes. A hardware semaphore is usually used to synchronize task execution while using shared resources which can be accessed by multiple cores.

Table 11 shows the functions that are used to synchronize tasks between the two cores when STM32CubeH7 is used. These functions are defined in stm32h7xx_hal_hsem.h.

Table 11. **HSEM functions**

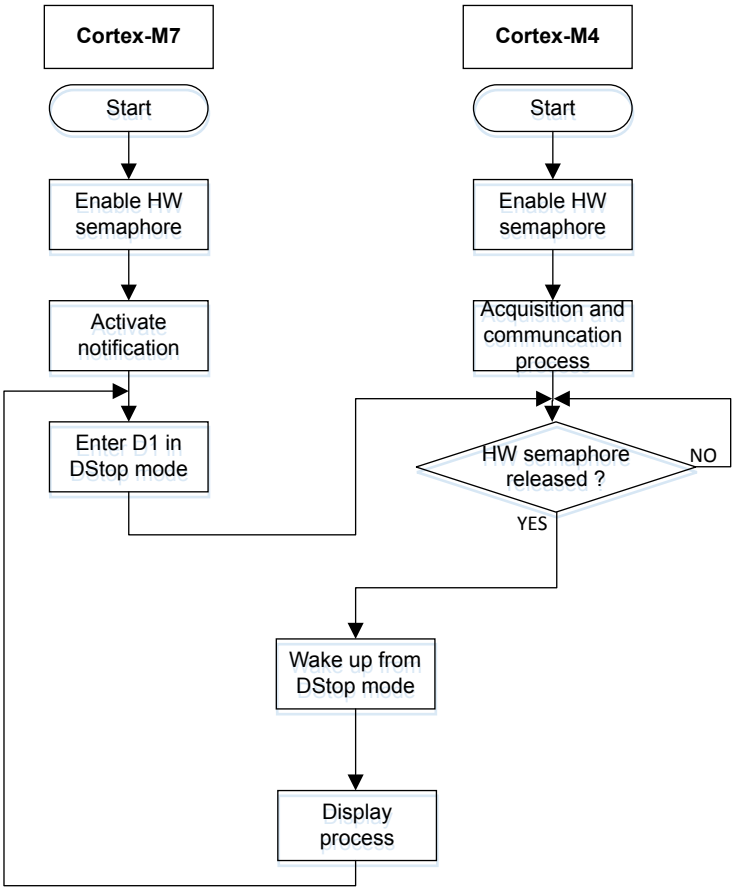
| Hardware semaphore (HSEM) | Description |
|---|---|
| `__HAL_RCC_HSEM_CLK_ENABLE()` | Enables the hardware semaphore. |

| Hardware semaphore (HSEM) | Description |
|---|---|
| `HAL_HSEM_ActivateNotification (SemMask )` | Activates HSEM notification to the Arm® Cortex®-M7 core. |
| `HAL_HSEM_FastTake( SemID )` | Uses by a CPU to take the hardware semaphore in order to inform the other CPU that it has completed its job. |
| `HAL_HSEM_Release( SemID , ProcessID )` | Releases the hardware semaphore when needed. |

Only one semaphore is used to synchronize the processing of both cores. The core synchronization mechanism is the following:

1. The two cores must first activate the semaphore separately by using `__HAL_RCC_HSEM_CLK_ENABLE( )`.
2. The Arm® Cortex®-M4 core then takes the HSEM using `HAL_HSEM_FastTake(SemID)` and starts processing its tasks in bare metal mode.
3. Once the task is compete, the core releases the semaphore to wake up the Arm® Cortex®-M7 by executing `HAL_HSEM_Release(SemID, ProcessID)`.
4. When the Arm® Cortex®-M7 has taken the semaphore, it executes its task using FreeRTOS™. When the task is compete, it informs the other core by releasing the semaphore. The following figure illustrates this sequence of operations.

**Figure 14. Core synchronization using a hardware semaphore**



# 4 Power consumption measurements

The measurements below have been obtained with the following settings and toolchain:

- Supply voltage = 3.3 V
- System clock (sys_ck) = 400 MHz
- EWARM 8.40 IAR workbench used with high size optimization level under typical temperature condition

Table 12 provides a summary of the power consumption measurements depending on the modes.

**Table 12. Power consumption measurement vs mode**

| Mode | Power consumption with LDO enabled | Power consumption with SMPS enabled | Unit |
|---|---|---|---|
| Mode 1 | 150 | 62.7 | mA |
| Mode 2 | 60 | 25.1 | |
| Mode 3 | 23 | 9.6 | |

# Revision history

Table 13. **Document revision history**

| Date | Version | Changes |
|---|---|---|
| 29-May-2019 | 1 | Initial release. |
| 29-Aug-2024 | 2 | Updated document title.<br>In the whole document, changed application example into application use case.<br>Updated Figure 7. Global flowchart of temperature acquisition depending on modes, Figure 8. Mode initialization sequence, Figure 9. Acquisition sequence, Figure 10. High-level block diagram of STM32H7x7 and DSI interface , Figure 12. Arm® Cortex®-M7 programming flowchart, Figure 13. Arm® Cortex®-M7 main task flowchart, and Figure 14. Core synchronization using a hardware semaphore.<br>Removed section *How to use the application example*.<br>Added EWARM toolchain in Section 4: Power consumption measurements. |

# Contents

# List of tables

# List of figures