AN5014
Application note

STM32H7x3 smart power management Expansion Package for STM32Cube

Introduction

STM32H7x3 microcontrollers deliver the maximum theoretical performance of the Arm® Cortex®-M7 core, regardless of whether code is executed from the embedded Flash memory or from an external memory: 2400 CoreMark® /1027 DMIPS at 480 MHz f_{CPU}.

STM32H7x3 devices embed a flexible power architecture. The system is split into three domains operating independently to optimize power efficiency.

This document is split into two parts:

- The first part describes the system supply configuration; how the CPU sub-system is configured by allocating peripherals to the CPU. It also describes the various operating modes, and how the different system partitions and clock activities are controlled in these modes.
- The second part provides an example of temperature acquisition based on I^2C transmission, using the X-NUCLEO-IKS01A2 expansion board (based on the STM32H7743ZI MCU). The purpose of this example is to highlight the smart power management of STM32Hx3 devices:
  - using three power domains
  - minimizing power consumption while keeping some activities running when needed (D3 Autonomous mode).

This application note is provided with the X-CUBE-PWRMGT-H7 Expansion Package that contains one project with four different workspaces:

- STM32H743ZI_Mode1: temperature acquisition with CPU in CSleep mode, D1 / D2 in DRun mode and D3 in Run mode
- STM32H743ZI_Mode2: temperature acquisition with CPU in CStop mode, D1 in DRun mode, D2 in DStandby mode and D3 in Run mode
- STM32H743ZI_Mode3: temperature acquisition with CPU in CStop mode, D1 and D2 in DStandby mode, and D3 in Run mode
- STM32H743ZI_Mode4: temperature acquisition with CPU in CStop mode, D1 and D2 in DStandby mode, and D3 switching between Run and Stop modes.

For further information on STM32H7x3 line devices, refer to the following documents available on *www.st.com*:

- STM32H7x3 advanced Arm®-based 32-bit MCUs reference manual (RM0433)
- STM32H743xx datasheet (DS12110) and STM32H753xx datasheet (DS12117)

Contents

List of tables

List of figures

1 System architecture

The advanced eNVM technology and architecture used on the STM32H7x3 line devices, enable the higher frequency of the core (up to 480 MHz). The multiple power domains provide a smart solution to the power-consumption challenges.

The system is partitioned as follows:

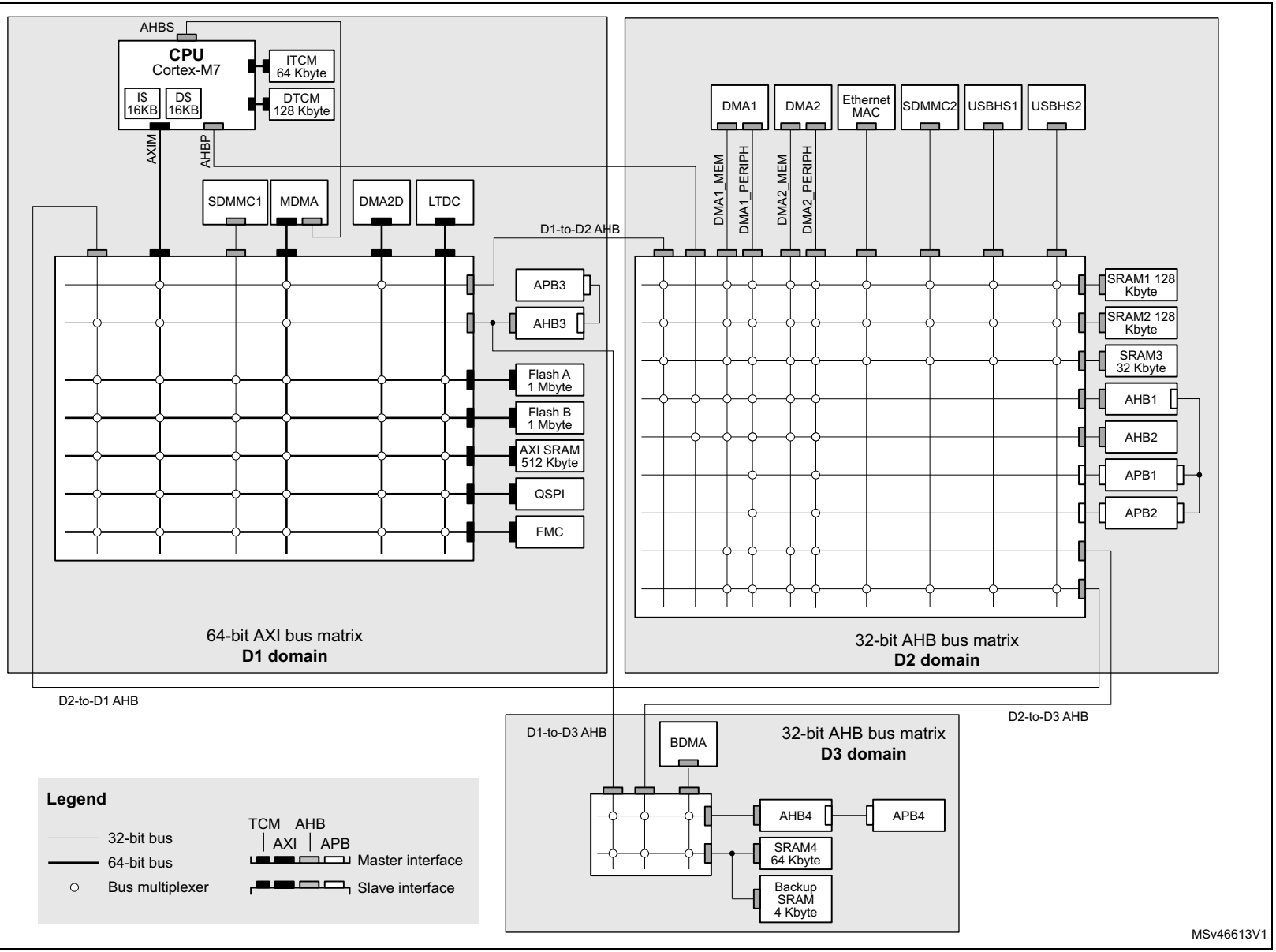
- one CPU sub-system: Arm®(a) Cortex®-M7 core, with associated peripherals clocked according to CPU activity
- Three power domains:
  - **D1 domain**: high-bandwidth and high-performance domain with the Cortex®-M7 core and acceleration mechanisms. This domain encompasses the high-bandwidth features and smart management thanks to the AXI bus matrix.
  - **D2 domain:** I/O processing domain, containing most peripherals that are less bandwidth demanding.
  - **D3 domain**: designed to manage the Low-power mode (embeds the system configuration block to keep the system state, the GPIO status). This domain is designed to be autonomous, embedding a 64-Kbyte RAM and a subset of peripherals to run basic functions, while the D1 and D2 domains can be shut-off to save power.

arm

Figure 1 illustrates the system architecture of the STM32H7x3 line devices.

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

Figure 1. System architecture for STM32H7x3 line devices



2 System supply configuration

The V_{CORE} domain supply can be provided by the voltage regulator (LDO) or by an external supply (through V_{CAP}: LDO bypass). V_{CORE} supplies all the digital circuitries except for the backup domain and the standby circuitry.

The V_{CORE} domain is split into the following sections:

- **D1 domain** containing the CPU (Cortex®-M7), Flash memory, memories interface and graphical peripherals
- **D2 domain** containing analog and digital peripherals
- **D3 domain** containing the system control, I/O logic and low-power peripherals.

The different supply configurations are controlled through the LDOEN and BYPASS bits in PWR control register 3 (PWR_CR3). The choice of the V_{CORE} domain supply can be done only once, before configuring the system clock source.

For more details about supply configuration control, refer to the power supplies section in the *STM32H7x3 advanced Arm®-based 32-bit MCUs* reference manual (RM0433).

2.1 Voltage regulator (LDO) supply

The voltage regulator (LDO) supplies the three power domains (D1, D2 and D3 - D1 and D2 can be switched off independently). The LDO output can be adjusted according to the application needs through the following power supply levels:

- **VOS0**:~1.35 V
- **VOS1**:~1.2 V
- **VOS2**: ~1.1 V
- **VOS3 / SVOS3**: ~1.0 V
- **SVOS4**: ~0.9 V
- **SVOS5**: ~0.7 V

Note: For more accurate values of voltage scaling output, refer to product datasheet.

According to the frequency, the voltage scaling (VOS) can be reduced to further optimize power consumption. By default, VOS3 is selected after power up. VOS can be changed on-the-fly in the PWR_D3CR register, to fit with the required system performance.

The voltage regulator provides three different operating modes: Main regulator (MR), Low-power (LP) or off. These modes are used depending on the system operating modes (Run, Stop and Standby).

The table below illustrates the different operating modes and voltage scaling of the voltage regulator in Run, Stop and Standby modes.

Table 1. Voltage regulator operating mode and voltage scaling

Mode	Regulator operating mode	VOS	CPU (MHz)	AXI (MHz)	AHB (MHz)	APB (MHz)	Description
Run	MR	VOS0	480	240	240	120	VCORE boost allowing to reach the system maximum frequency
		VOS1	400	200	200	100	High performance
		VOS2	300	150	150	75	Medium performance and consumption
		VOS3	200	100	100	50	Optimized performance and low-power consumption. By default VOS3 is selected after system reset.
Stop	MR or LP	SVOS3	NA				The regulator can be: – in Main mode to allow fast exit from Stop mode – in LP mode to obtain a lower V_{CORE} supply level and extend the exit-from-stop latency.
	LP	SVOS4					The Stop mode consumption is further reduced with a lower voltage level for SVOS4 and SVOS5 scaling. In SVOS4/SVOS5, the content of the registers and memories is not guaranteed.
		SVOS5					
Standby	Off	NA					Content of the registers and memories is lost except for the Standby circuitry and the backup domain.

The functions defined in stm32h7xx_hal_pwr_ex.c file and described in the table below, allow the voltage scaling configuration in STM32Cube_FW_H7.

Table 2. Configure the voltage scaling using STM32Cube_FW_H7

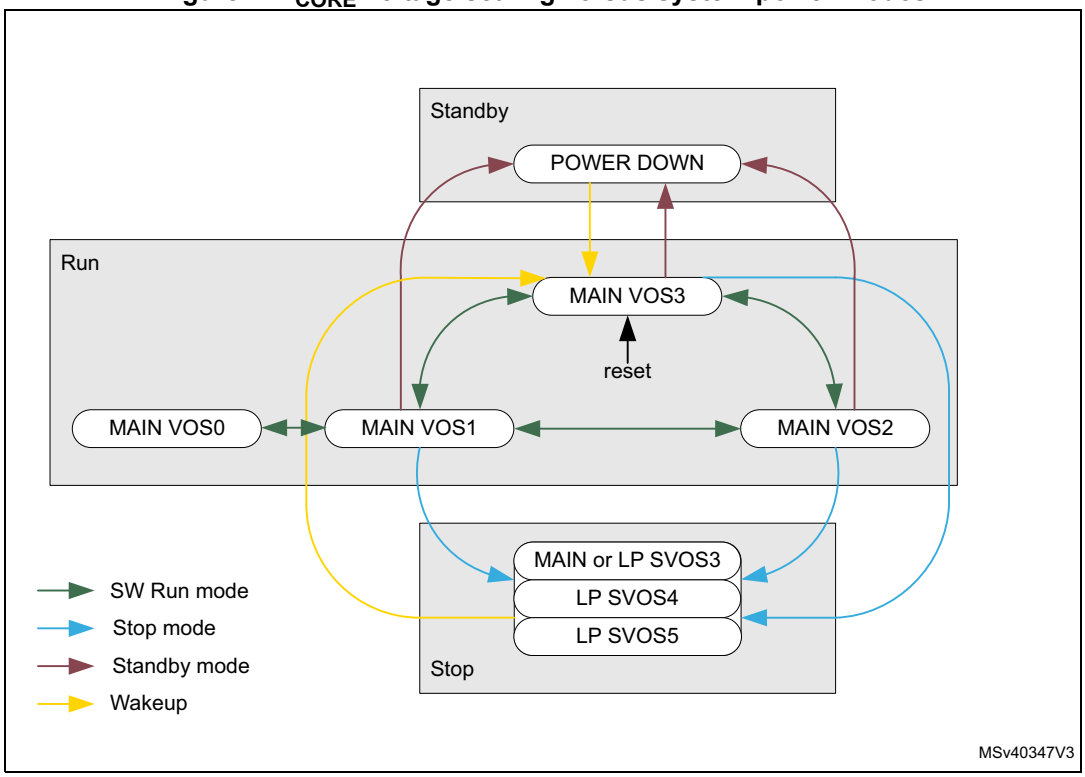
Function	Ouptut	Input	Description
HAL_PWREx_Control VoltageScaling	HAL_Status TypeDef	PWR_REGULATOR_VOLTAGE_SCALE0 PWR_REGULATOR_VOLTAGE_SCALE1 PWR_REGULATOR_VOLTAGE_SCALE2 PWR_REGULATOR_VOLTAGE_SCALE3	Configure the output voltage of the main internal regulator.
HAL_PWREx_Control StopMode VoltageScaling		PWR_REGULATOR_SVOS_SCALE3 PWR_REGULATOR_SVOS_SCALE4 PWR_REGULATOR_SVOS_SCALE5	Configure the output voltage of the main internal regulator in Stop mode.

The Stop mode voltage scaling for SVOS4 and SVOS5 sets the voltage regulator in Low-power (LP) mode automatically to further reduce power consumption.

When preselecting SVOS3, the use of the voltage regulator Low-power mode can be selected by LPDS register bit in the PWR_CR1 register.

The figure below illustrates the V_{CORE} voltage scaling versus system power modes.

Figure 2. V_{CORE} voltage scaling versus system power modes

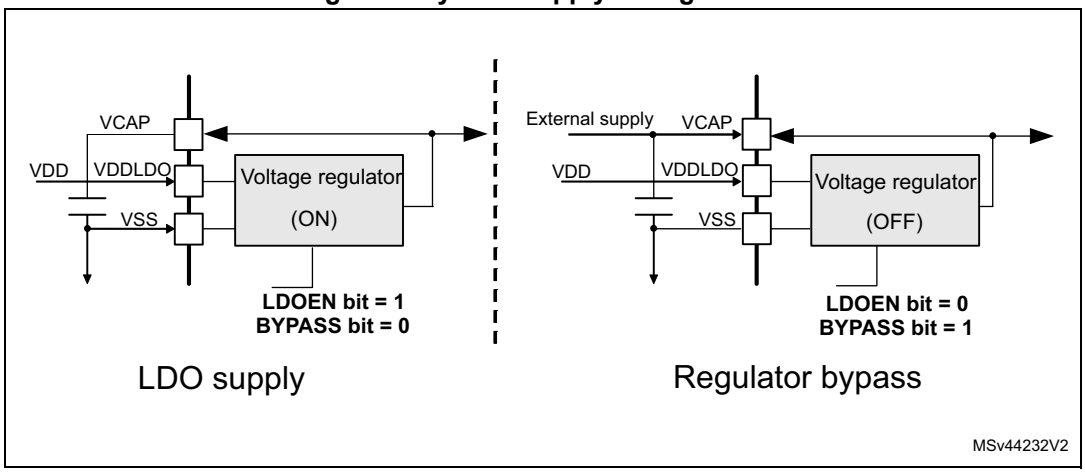


2.2 Voltage regulator bypass

In STM32H7x3 line devices, the regulator bypass is managed by software through LDOEN and BYPASS bits in the PWR_CR3 register. Vcore is delivered by an external power supply through VCAP pin. At system startup, the MCU starts under LDO then the user can switch to the regulator bypass mode by modifying the LDOEN and BYPASS bits with the software.

The figure below illustrates the system supply configuration for STM32H7x3 line devices (refer to the 'VCAP external capacitor' section of the datasheet for external capacitor values).

Figure 3. System supply configuration



In STM32Cube_FW_H7, the function described in the table below allows the usage of bypass voltage regulator feature.

Table 3. Configure the system power supply using STM32Cube_FW_H7

Function	Output	Input	Description
HAL_PWREx_ConfigSupply	HAL_StatusTypeDef	PWR_LDO_SUPPLY	The LDO supplies the V_{core} power domains.
		PWR_EXTERNAL_SOURCE_SUPPLY	The LDO is bypassed.

3 CPU sub-system configuration

3.1 Peripheral allocation

The peripheral allocation is used by the following peripherals:

- the RCC to automatically control the clock gating according to the CPU and domain modes
- the PWR to control the supply voltages of D1, D2 and D3 domains

The CPU allocates a peripheral by setting the dedicated PERxEN bit located in the RCC_xxxxENR or RCC_C1_xxxxENR registers.

When using STM32Cube_FW_H7, the peripheral is allocated through the dedicated function defined in stm32h7xx_hal_rcc.h:

```
__HAL_RCC_xxxx_CLK_ENABLE();
```

The CPU controls also the peripheral clocks gating when it is in CSleep mode through PERxLPEN bits located into RCC_xxxxLPENR or RCC_C1_xxxxLPENR registers.

When using STM32Cube_FW_H7, the peripheral clocks are controlled in CSleep mode through the dedicated function defined in stm32h7xx_hal_rcc.h:

```
__HAL_RCC_xxxx_CLK_SLEEP_ENABLE();
```

The peripheral allocation bits (PERxEN bits) are used by the hardware to provide the kernel and bus interface clocks to the peripherals. However they are also used to link peripherals to the CPU (CPU sub-system). In this way, the hardware is able to safely gate the peripheral clocks and bus matrix clocks according to CPU states. The PWR block also uses this information to control properly the domain states.

Figure 4 gives an example of peripheral allocation.

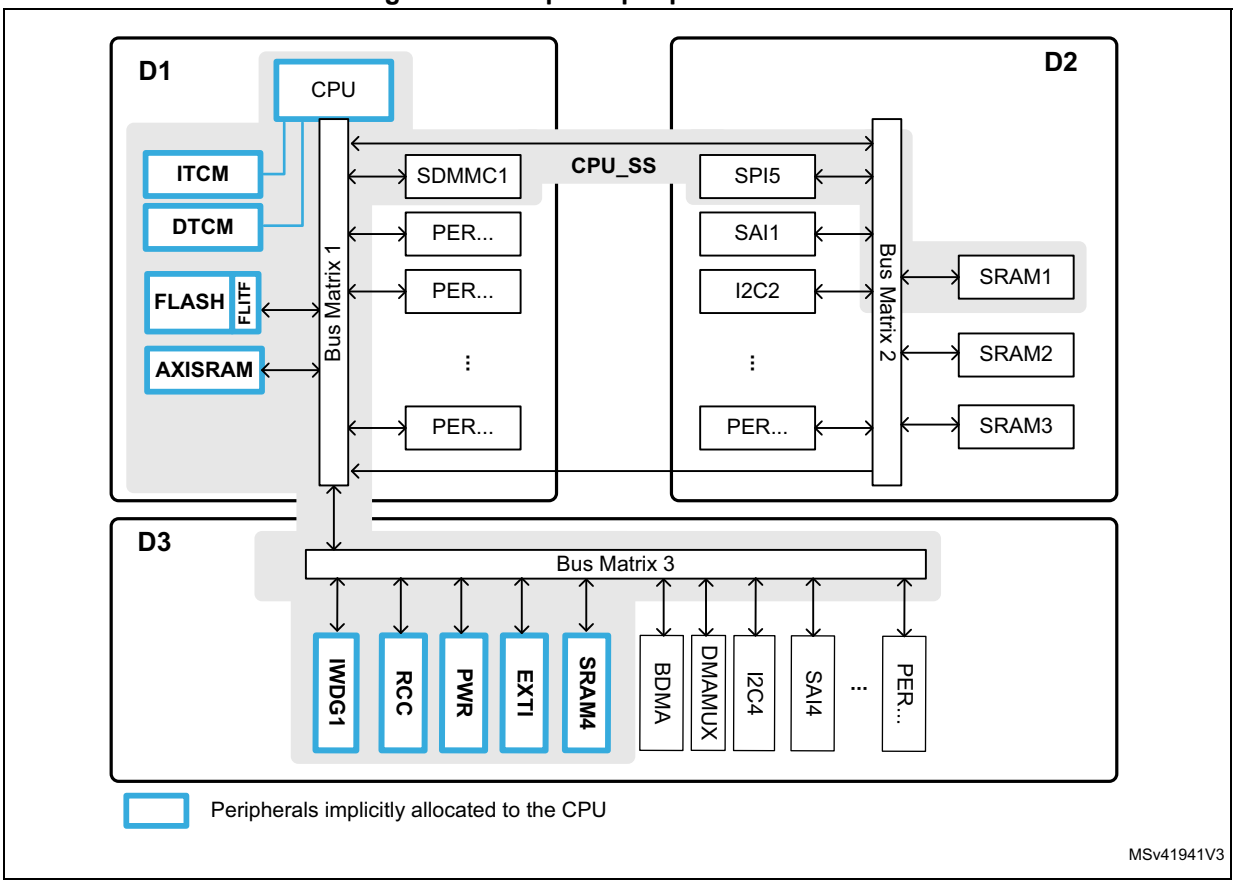
The CPU enables SDMMC1, SPI5 and SRAM1. Other memories (FLASH, AXISRAM, ITCM, DTCM and SRAM4) are implicitly allocated to the CPU. As a result, there is no enable bit allowing the CPU to allocate these memories.

The CPU needs to enable the memories located in the D2 domain (SRAM1, SRAM2 and SRAM3) before using them. The BackupSRAM has a dedicated enable bit in order to gate the bus interface clock. The CPU needs to enable the BackupSRAM prior to use it.

The group composed of the CPU, bus matrix 1/2/3 and allocated peripherals, makes up a **CPU-subsystem**.

Note: FLASH, AXISRAM, ITCM, DTCM, SRAM4, IWGD1, IWGD2, PWR, EXTI and RCC are common resources, implicitly allocated to the CPU.

Figure 4. Example of peripheral allocation



3.2 D3 Autonomous mode

The Autonomous mode allows providing the peripheral clocks to peripherals located in the D3 domain, even if the CPU is in CStop mode. When a peripheral is enabled and has its autonomous bit activated (PERxAMEN bit in the RCC_D3AMR register), this peripheral receives its clocks to the D3 domain state, if the CPU is in CStop mode.

When using STM32Cube_FW_H7, the peripheral Autonomous mode is enabled/disabled via the following functions defined in stm32h7xx_hal_rcc.h file:

- Enable peripheral in Autonomous mode:

  ```
  __HAL_RCC_xxxx_CLKAM_ENABLE();
  ```

- Disable peripheral from Autonomous mode:

  ```
  __HAL_RCC_xxxx_CLKAM_DISABLE();
  ```

Setting the RUN_D3 bit in the PWR_CPUCR register, keeps the D3 domain in Run mode while the CPU is in CStop mode and D1 and D2 domains are in DStop or DStandby mode.

When using STM32Cube_FW_H7, the system D3 domain is kept in Run mode regardless the CPU subsystem modes, via the function defined in stm32h7xx_hal_pwr_ex.c and described in the table below.

Table 4. Configure the D3 domain state regardless CPU subsystem modes using STM32Cube_FW_H7

Function	Output	Input	Description
HAL_PWREx_Config D3Domain	Void	PWR_D3_DOMAIN_RUN	D3 domain stays in Run mode regardless of the CPU sub-system mode.
		PWR_D3_DOMAIN_STOP	D3 domain follows the CPU sub-system mode.

Note: RCC does not switch off the PLLs as the D3 domain in Run mode.

4 Power control modes

4.1 Operating modes

The operating modes allow controlling the clock distribution to the different system blocks and powering them. The system operating mode is driven by the CPU subsystem, D2 domain and system D3 autonomous wakeup. The CPU subsystem includes multiple domains depending on its peripheral allocation.

The operating modes available for the different system blocks are detailed in the table below.

Table 5. STM32H7x3 line operating modes

CPU subsystem and domains	Mode	Description
CPU subsystem	CRun	– CPU and CPU subsystem peripheral(s) allocated via RCC PERxEN bits in RCC_C1_xxxxENR or RCC_xxxxENR, are clocked.
	CSleep	– The CPU clock is stalled and the CPU subsystem allocated peripheral(s) clock operates according to RCC PERxLPEN bits.
	CStop	– CPU and CPU subsystem peripheral(s) clocks are stalled.
D1 domain	DRun	– The D1 domain bus matrix is clocked. – The CPU subsystem operates in CRun or CSleep mode.
	DStop	– The D1 domain bus matrix clock is stalled. – The CPU subsystem operates in CStop mode and the **PDDS_D1** bit selects DStop mode.
	DStandby	– The D1 domain Vcore supply is powered off. – The CPU subsystem operates in CStop mode and the PDDS_D1 bit selects DStandby mode.
D2 domain	DRun	– The D2 domain bus matrix clock is clocked. – The CPU subsystem has an allocated peripheral in the D2 domain and the CPU subsystem operates in CRun or CSleep mode.
	DStop	– The D2 domain bus matrix clock is stalled. – The CPU subsystem has no peripheral allocated in the D2 domain and **PDDS_D2** bit selects DStop mode. – The CPU subsystem has an allocated peripheral in D2 domain. The CPU subsystem operates in **CStop** mode and **PDDS_D2** bit selects DStop mode.
	DStandby	– The D2 domain Vcore supply is powered off. – The CPU subsystem has no peripherals allocated in the D2 domain and **PDDS_D2** bit selects DStandby mode. – The CPU subsystem has an allocated peripheral in the D2 domain. The CPU subsystem operates in CStop mode and **PDDS_D2** bit selects DStandby mode.

Table 5. STM32H7x3 line operating modes (continued)

CPU subsystem and domains	Mode	Description
System/ D3 domain	Run	– The system clock and D3 domain bus matrix clock are running. – The CPU subsystem is in CRun or CSleep mode. – A wakeup signal is active (system D3 Autonomous mode).
	Stop	– The system clock and D3 domain bus matrix clock are stalled. – The CPU subsystem is in CStop mode **and** all wakeup signals are inactive(1) **and** at least one PDDS_Dn bit for any domain selects Stop mode.
	Standby	– The system is powered down. – The CPU subsystem is in CStop mode **and** all wakeup signals are inactive(1) **and** all PDDS_Dn bits for all domains select Standby mode

1. wakeup signals are enabled but no wakeup event is pending.

The operating modes for D1 and D2 domains are independent. However the system operating mode depends on D3 domain operating mode. The table below details the system state according to D1 and D2 domains states.

Table 6. System states for D1 / D2 domains states

D1 domain power mode	D2 domain power mode	System/D3 domain power mode
DRun	DRun/DStop/DStandby	Run
DStop/DStandby	DStop/DStandby	
DStop/DStandby	DStop/DStandby	Stop
DStandby	DStandby	Standby

The D1, D2 and system/D3 domains are supplied from a single regulator at a common V_{CORE} level. The V_{CORE} supply level follows the system operating mode (Run, Stop, and Standby). The D1 domain and/or D2 domain supply can be powered down individually when these domains are in DStandby mode.

When the system is in RUN mode, the D2 domain is by default in DStop mode unless the CPU allocates at least one peripheral belonging to this domain.

.The table below describes the D2 domain states according to CPU peripheral allocation and D1 domain states.

Table 7. D2 domain states overview

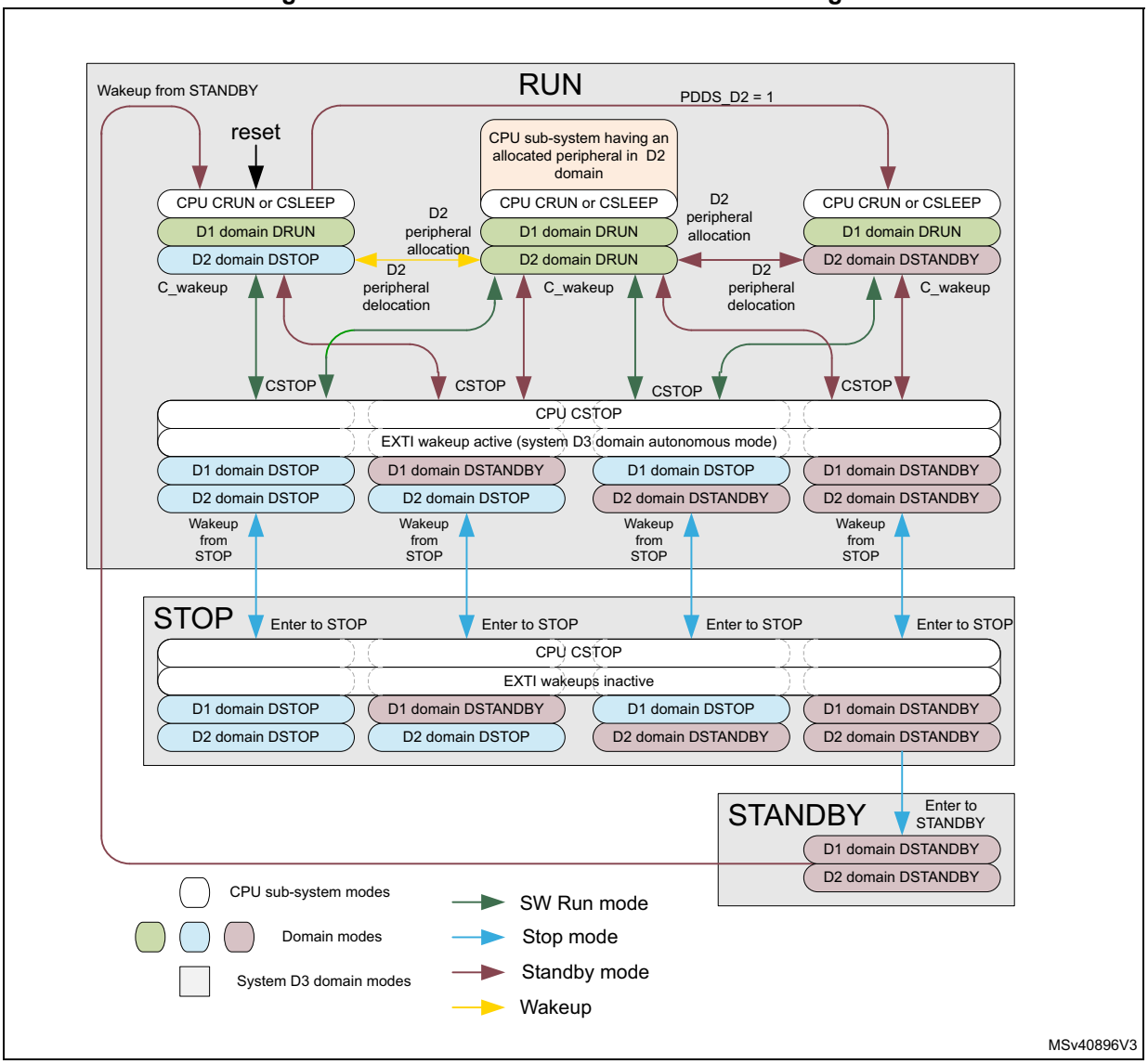
Peripheral allocation in D2 domain	CPU	D1 domain	D2 domain
None	CRun or CSleep	DRun	DStop / DStandby
	CStop	DStop / DStandby	

Table 7. D2 domain states overview (continued)

Peripheral allocation in D2 domain	CPU	D1 domain	D2 domain
Peripherals allocated	CRun or CSleep	DRun	DRun
	CStop	DStop / DStandby	DStop / DStandby

The state diagram below illustrates the different power control modes available for STM32H7x3 devices.

Figure 5. Power control modes detailed state diagram



4.2 Low-power modes

Several low-power modes are available to save power when the CPU does not execute code (when waiting for an external event). The user must select the mode providing the best compromise between low power-consumption, short startup-time and available wakeup sources for the application.

STM32H7x3 line devices feature the following low-power modes:

- CSleep (CPU clock stopped)
- CStop (CPU sub-system clock stopped)
- DStop (domain bus matrix clock stopped)
- Stop (system clock stopped)
- DStandby (domain powered down)
- Standby (system powered down)

When using STM32Cube_FW_H7, several functions are defined for low-power modes. These functions are described in the table below.

Table 8. Functions description used for low-power modes in STM32Cube_FW_H7

Low-power mode	Function	Input	Description
CSleep	HAL_PWR_Enter SLEEPMode	PWR_MAINREGULATOR_ON PWR_LOWPOWERREGULATOR_ON	The regulator parameter is not used for STM32H7x3 line devices and is kept as parameter just to maintain compatibility with the lower-power families (STM32L).
		PWR_SLEEPENTRY_WFI PWR_SLEEPENTRY_WFE	Specifies if Sleep mode is entered with WFI or WFE instruction.
CStop / DStop	HAL_PWREx_Enter STOPMode	PWR_MAINREGULATOR_ON PWR_LOWPOWERREGULATOR_ON	Specifies the regulator state in Stop mode.
		PWR_STOPENTRY_WFI PWR_STOPENTRY_WFE	Specifies if Stop mode is entered with WFI or WFE instruction.
		PWR_D1_DOMAIN PWR_D2_DOMAIN PWR_D3_DOMAIN	Specifies the domain to enter Stop mode.
Stop	HAL_PWR_Enter STOPMode	PWR_MAINREGULATOR_ON PWR_LOWPOWERREGULATOR_ON	Specifies the regulator state in Stop mode.
		PWR_STOPENTRY_WFI PWR_STOPENTRY_WFE	Specifies if Stop mode is entered with WFI or WFE instruction.
DStandby	HAL_PWREx_Enter STANDBYMode	PWR_D1_DOMAIN PWR_D2_DOMAIN PWR_D3_DOMAIN	Specifies the domain to enter Standby mode.
Standby	HAL_PWR_Enter STANDBYMode	Void	System enters Standby mode.

5 Operating modes comparison between STM32F7 Series and STM32H7x3 line devices

What is new in STM32H7x3 line devices regarding STM32F7 Series MCUs, is the flexibility of managing power supply on three domains. For example you can switch off a whole domain and keep the two others running.

The table below illustrates shared operating modes between STM32F7 Series and STM32H7x3 devices.

Table 9. Shared operating modes - STM32F7 Series and STM32H7x3 line devices

Shared operating modes	STM32F7 Series devices	STM32H7x3 line devices			
		CPU state	D1 domain	D2 domain	D3 domain
Run	Run	CRun	DRun	DRun	Run
Sleep	Sleep	CSleep	DRun	DRun	Run
Stop	Stop	CStop	DStop	DStop	Stop
Standby	Standby	CStop	DStandby	DStandby	Standby

To use low-power modes, STM32F7 Series and STM32H7x3 line devices share also the same functions defined in STM32Cube_FW_F7 and STM32Cube_FW_H7. These functions are described in the table below.

Table 10. Shared functions - STM32F7 Series and STM32H7x3 line devices

Shared low-power mode	Function	Output	Input	Description
Sleep	HAL_PWR_EnterSLEEPMode	Void	Regulator(1) SLEEPEntry	Cortex®-M7 enters Sleep mode.
Stop	HAL_PWR_EnterSTOPMode		Regulator STOPEntry	System enters Stop mode.
Standby	HAL_PWR_EnterSTANDBYMode		Void	System enters Standby mode.

1. This parameter is not used for STM32H7x3 devices nor for STM32F7 Series. It is kept as a parameter just to maintain compatibility with the lower-power MCUs.

6 Peripheral clock distribution

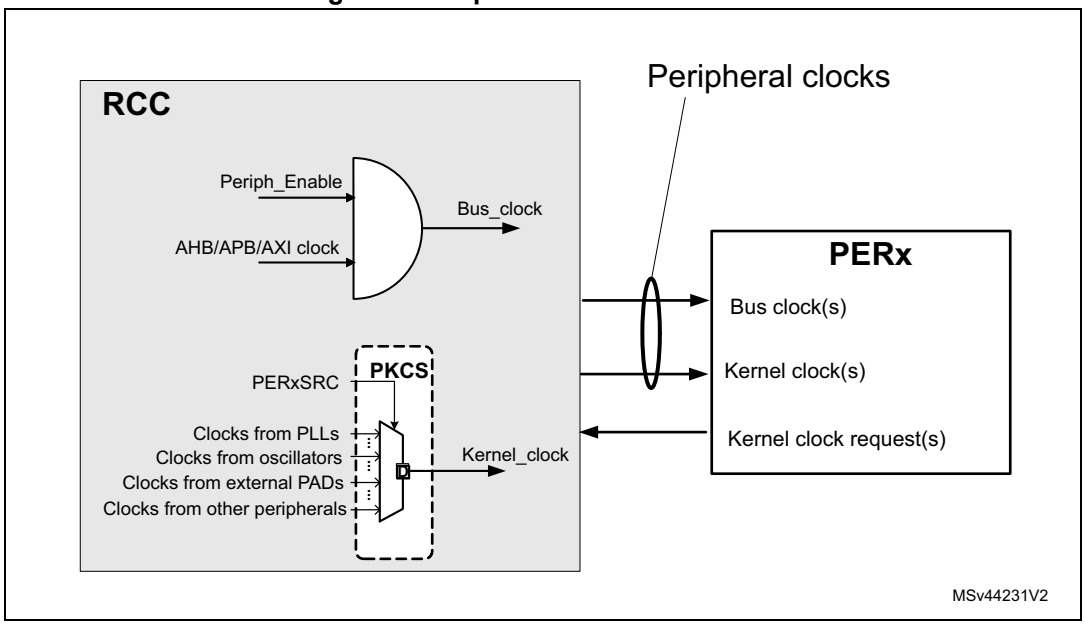
The peripheral clocks are provided by the RCC to the peripherals. Two kinds of clock are available:

- bus interface clocks
- kernel clocks.

On STM32H7x3 line devices, the peripherals generally receive one or several bus and kernel clocks.

The figure below describes the peripheral clock distribution on STM32H7x3 line devices.

Figure 6. Peripheral clock distribution



A kernel clock allows the peripheral to use a different clock frequency for the peripheral function (compared to the peripheral bus clock).

Peripherals with a kernel clock are classified in several groups:

- **PK1**: Peripherals with a kernel clock following the same clock gating as the peripheral bus clock.

  Examples: FMC, QUADSPI, DFSDM1, I2Cx, LPTIMx, SPI4/5/6, SWPMI, USARTx, UARTx, LPUART1, SDMMC1/2, SAIx, ADCx, DSI, FDCAN, HDMI-CEC, LTDC, RNG, SPDIFRX, USBxLPI, USBxOTG.

- **PK2**: Peripherals with a kernel clock request signal (select clock source HSI or CSI).

  Examples: I2Cx, SPI4/5/6, USARTx, UARTx, LPUART1.

  These PK2 peripherals are able to run when the CPU subsystem allocating the peripheral is in CStop or the domain is in DStop or the whole system is in Stop.For example, if the system is expecting messages via I2C4, the whole system can be put in Stop mode. When the I2C4 peripheral detects a START bit, it generates a kernel clock

request which enables the HSI or CSI. This kernel clock is provided only to the requester (I2C4) and decodes then the incoming message.

- **PK3**: peripherals with kernel clock source LSE or LSI.

  Examples: HDMI-CEC, LPTIM1/2/3/4/5, RNG, RTC/AWU, USARTx, UARTx, LPUART1.

  These PK3 peripherals are able to run when the CPU subsystem allocating the peripheral is in CStop, when the domain is in DStop or when the system is in Stop.

- **PK4**: peripherals with external slave clock as kernel clock.

  Examples: MDIOS, I2Cx, SPIx.

The peripherals having a kernel clock request, operates in Stop mode if the two following conditions are filled:

- the system uses SVOS3 level.
- the PERxAMEN bit, when available on the peripheral (ones located in D3 domain), is set to request the kernel clock.

When using STM32Cube_FW_H7, the kernel clock of the peripheral is selected using the following function defined in stm32h7xx_hal_rcc_ex.h file:

```
__HAL_RCC_xxxx_CONFIG (__xxxCLKSource__);
```

For more details about kernel clock distribution and peripheral clock gating, refer to the *STM32H7x3 advanced Arm®-based 32-bit MCUs* reference manual (RM0433).

7 Wakeup from Stop/CStop mode

The extended interrupt and event controller module (EXTI) allows the system and/or the CPU to wake up from respectively the Stop mode or the CStop mode.

What is new in STM32H7x3 devices is that these events are coming from peripherals. These events are handled by the EXTI as configurable or as direct events.

There are three types of peripheral output signals connected to the EXTI input events:

- **Wakeup signals**: generated by the peripheral without any bus- interface clock (when the only clock running is the kernel one). These signals are referred to as xxx_wkup and are not available for some peripherals.

Note: the use of a kernel clock for I2C and SPI peripherals is supported only in Slave mode, so the wakeup signals available in these peripherals are supported only in Slave mode.

- **Interrupt signals**: generated only if the peripheral bus-interface clock is running. These interrupt signals are generally connected directly to NVIC of CPU and referred to as xxx_it.
- **Signals**: pulses generated by the peripheral. Once a peripheral generates a signal, no action (flag clearing) is required at peripheral level.

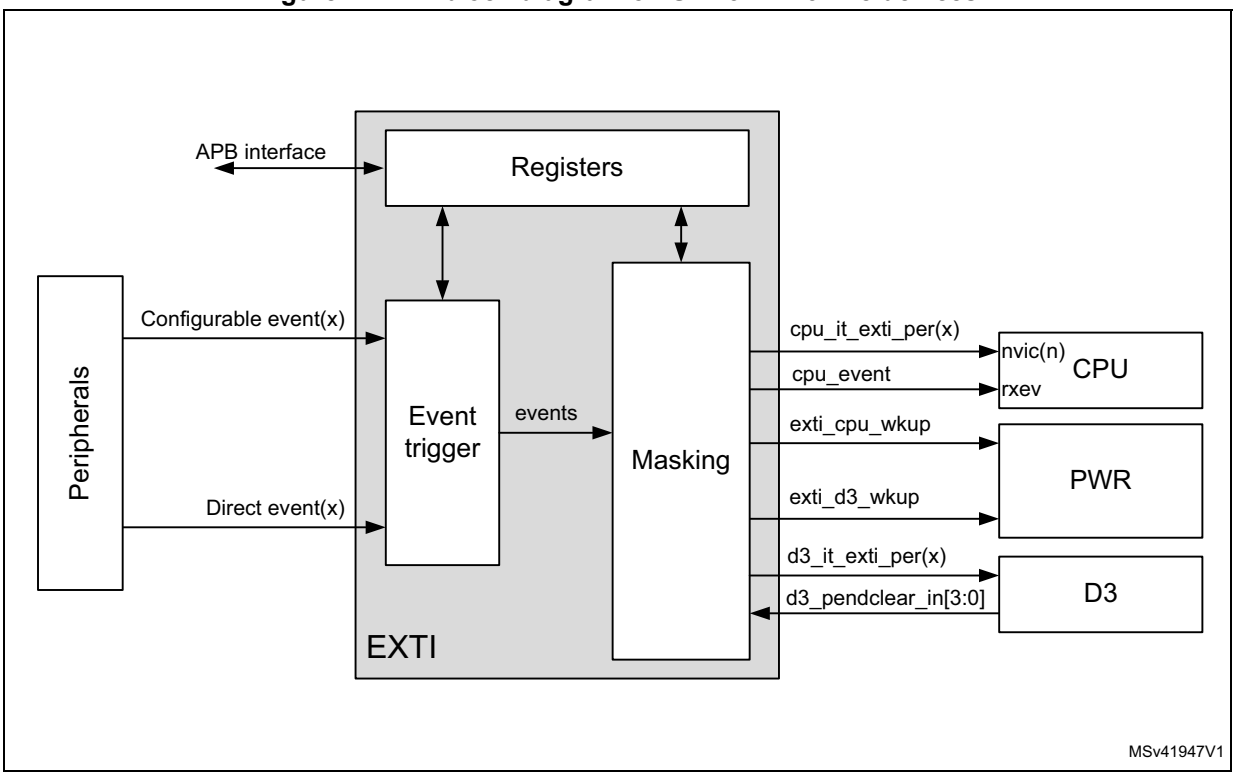
Each EXTI input event has different wakeup capability or possible target:

- **CPU wakeup (CPU)**: the input event is enabled to wake up the CPU.
- **CPU and D3 domain wakeup for Autonomous Run mode (ANY)**: the input event is enabled to wake up the CPU or the D3 domain only for Autonomous Run mode phase.

For more details about the EXTI, refer to the reference manual RM0433: dedicated section for EXTI main features and specific table for wakeup inputs.

The figure below illustrates the EXTI bloc diagram on the STM32H7x3 line devices.

Figure 7. EXTI block diagram on STM32H7x3 line devices



EXTI pending requests clear for D3 Run mode inputs

As described above, the EXTI event inputs are able to wake up the D3 domain for Autonomous Run mode which has a pending request logic.

This pending request can be cleared by four input sources, allowing D3 domain to resume the Stop mode.

- dmamux2_evt6
- dmamux2_evt7
- lptim4_out

The figure and table below describe the D3 pending request clear logic and the EXTI pending requests clear inputs.

Figure 8. D3 pending request clear logic

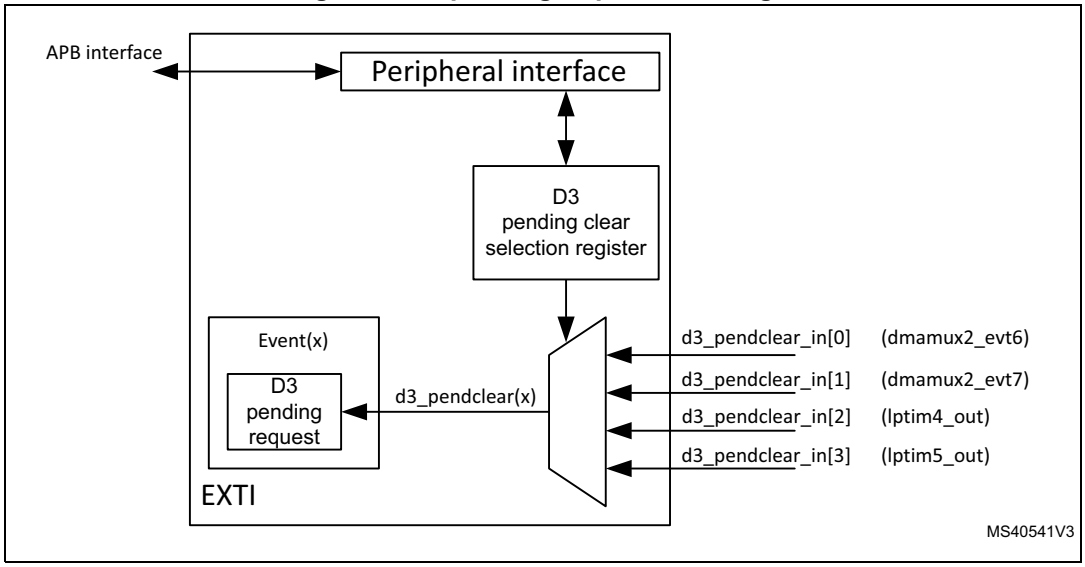


Table 11. EXTI pending request clear inputs

Source				Destination			
Domain	**Bus**	**Peripheral**	**Signal**	**Signal**	**Peripheral**	**Bus**	**Domain**
D3	AHB4	DMAMUX2	dmamux2_evt6	PRC0	EXTI	APB4	D3
			dmamux2_evt7	PRC1			
	APB4	LPTIM4	lptim4_out	PRC2			
		LPTIM5	lptim5_out	PRC3			

In STM32Cube_FW_H7, the function described in the table below allows the configuration of the EXTI input line for D3 domain, in order to specify the clear source of D3 pending event.

Table 12. Specifying the clear source of D3 pending event in STM32Cube_FW_H7

Function	Output	Input	Description
HAL_EXTI_D3_EventInputConfig	Void	EXTI_Line	Specifies the EXTI line that is waking up D3 domain.
		EXTI_LineCmd: Enable/disable	Enable or disable the EXTI line.
		EXTI_ClearSrc: – BDMA_CH6_CLEAR – BDMA_CH7_CLEAR – LPTIM4_OUT_CLEAR – LPTIM5_OUT_CLEAR	This parameter specifies the clear source of D3 pending event to allow D3 resume Stop mode.

8 Low-power application

8.1 Low-power application example

This example is based on I^2C transmission using ST shield (X-NUCLEO-IKS01A2). The purpose is to highlight the smart power management of STM32H7x3 devices using three power domains.

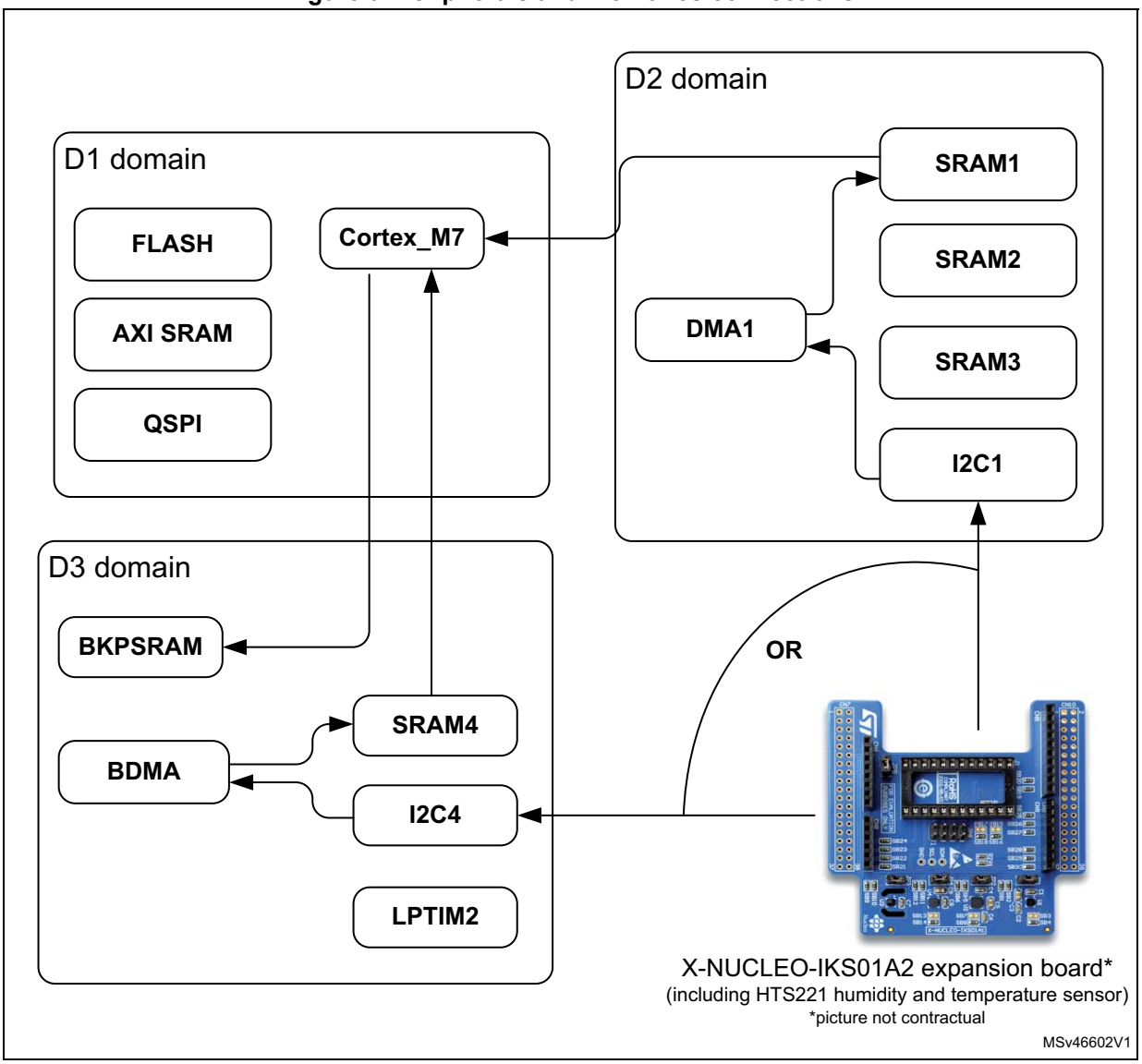
The demo code is based on four modes:

- **Mode 1**: this mode is equivalent to legacy products which keep all domains active to transfer data from ST shield through I2C1 (with CPU in Sleep mode).
- **Mode 2:** in this mode, the power efficiency is enhanced keeping D1 in DRun, D3 in Run and D2 in DStandby. The data transfer is switched to I2C4 instead of I2C1. This highlights the architecture flexibility, moving from one peripheral instances in D2 to another in D3, without changing external connections.
- **Mode 3**: same as Mode 2 but the D1 domain is in DStandby for more power- energy saving. This mode shows the D3 Autonomous Run mode.
- **Mode 4:** same as Mode 3 but the D3 domain switches between two modes (Run and Stop) all along the transfer, in order to optimize power-consumption.

The demo code uses the Nucleo board NUCLEO-H743ZI and the X-NUCLEO-IKS01A2 expansion board to get temperature values from HTS221 sensor via I^2C bus.

The figure below shows the connections between peripherals and memories.

Figure 9. Peripherals and memories connections



8.1.1 Mode 1

This mode is equivalent to Sleep mode in STM32F7 Series devices.

- Temperature acquisition from shield to SRAM1 through I2C1 in the D2 domain
- Saving data from SRAM4 to BackupSRAM after waking up Cortex®-M7

Table 13. Mode 1 - operating modes for each domain

D1 domain	D2 domain	D3 domain
DRun (CSleep)	DRun	Run

8.1.2 Mode 2

- Temperature acquisition from shield to SRAM4 through I2C4 in the D3 domain
- Saving data from SRAM4 to BackupSRAM after waking up Cortex®-M7

Table 14. Mode 2 - operating modes for each domain

D1 domain	D2 domain	D3 domain
DRun (CSleep)	DStandby	Run

8.1.3 Mode 3

- Temperature acquisition from shield to SRAM4 through I2C4 in the D3 domain
- Saving data from SRAM4 to BackupSRAM after waking up Cortex®-M7

Table 15. Mode 3 - operating modes for each domain

D1 domain	D2 domain	D3 domain
DStandby	DStandby	Run

8.1.4 Mode 4

- Temperature acquisition from shield to SRAM4 through I2C4 in the D3 domain
- Saving data from SRAM4 to BackupSRAM after waking up Cortex®-M7

Table 16. Mode 4 - operating modes for each domain

D1 domain	D2 domain	D3 domain
DStandby	DStandby	Run / Stop

8.2 Mode 4 - D3 Autonomous mode

This section details the D3 Autonomous mode (Mode 4) and its benefits on power consumption.

8.2.1 Memory retention

The D3 domain features 64 Kbytes of SRAM (SRAM4), available to retain data while the D1 and D2 domains are in DStandby mode.

This feature can be used in several use-cases:

- To retain the application code in order to recover properly from DStandby mode
- To retain the data from / to a sensor when the CPU enters CStop (with D1 or D2 domain in DStandby) between two consecutive operations.

SRAM4 remains available as long as the system is not in Standby mode. But if the system is in Standby mode, it is still possible to retain data in BackupSRAM. Its size is optimized to 4 Kbytes to reduce the associated leakage.

In the given example, both the SRAM4 and BKUP_SRAM are used, respectively for saving the transmitted data from temperature sensor when D3 is in Autonomous mode, and saving the final data after waking up the CPU from CStop mode.

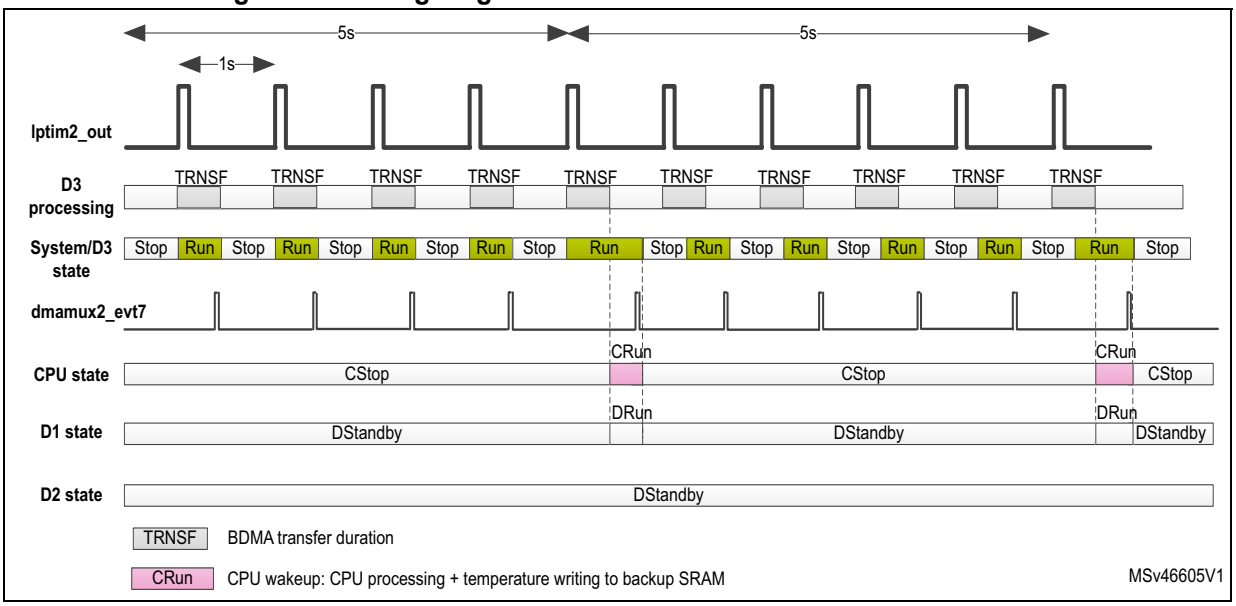
8.2.2 Example description

Figure 10 shows the proposed implementation. At regular time interval given by LPTIM2 (each 1 s), the D3 domain wakes up from Stop mode. The D3 domain executes the following tasks in Autonomous mode:

1. transfer the data from SRAM4 to I2C4 using BDMA
2. when the I2C4 interface indicates that the last 2 bytes has been transferred from I2C4 to SRAM4, the D3 domain is switched to Stop mode.

After transferring the first or last 10 bytes, BDMA_channel7 (BDMA_ch7) sends an interrupt to wake up the CPU from CStop mode. When the CPU is in CRun mode, it prepares data located into SRAM4 to be processed and transferred to the BackupSRAM.

Figure 10. Timing diagram of SRAM4 to I2C4 transfer with BDMA



Note: To toggle the D3 domain between Stop and Run modes, the Run mode must be triggered by wakeup event (lptim2_out each 1 s) so that the D3 domain can clear this event with dmamux2_evt7 signal and switch back the D3 domain to Stop mode.

RCC programming

In this example, the CPU sub-system also includes the peripherals of the D3 domain that are used for data transfer: LPTIM2, BDMA, DMAMUX2, SRAM4, I2C4, BackupSRAM.

These peripherals must be programmed in Autonomous mode, in order to operate even when the CPU is in CStop mode.

When using STM32Cube_FW_H7, the following functions defined in stm32h7xx_hal_rcc.h file are used to program peripherals in Autonomous mode:

- `__HAL_RCC_LPTIM2_CLKAM_ENABLE();`
- `__HAL_RCC_BDMA_CLKAM_ENABLE();`
- `__HAL_RCC_D3SRAM1_CLKAM_ENABLE();`
- `__HAL_RCC_I2C4_CLKAM_ENABLE();`
- `__HAL_RCC_BKPRAM_CLKAM_ENABLE();`

PWR programming

In this mode, the PWR block must be programmed in order to:

- prevent system D3 domain to enter Standby mode when the data transfer is complete
- allow D1 and D2 domains to enter DStandby mode
- define the working voltage according to system modes (SVOS3).

EXTI programming

The lptim2_out signal is used to wake up the D3 domain from Stop mode when LPTIM2 time interval has elapsed (each 1 s). The lptim2_out signal is connected to the EXTI input event line number 49.

Selecting BDMA_ch7 as pendclear source of the D3 domain, allow to switch back the D3 domain to Stop mode.

When using STM32Cube_FW_H7, the following function defined in stm32h7xx_hal.c is used to select lptim2_out signal as wakeup source and dmamux2_evt7 as pendclear source:

```
__HAL_EXTI_D3_EventInputConfig(EXTI_LINE49,ENABLE,BDMA_CH7_CLEAR);
```

BDMA and DMAMUX programming

Four BDMA channels are required to execute data transfers between SRAM4 and I2C4:

- **BDMA_channel0 (BDMA_ch0)** is used to transfer data from SRAM4 to I2C4_CR2 register, in order to configure the I2C4 interface to request write transfer to HTS221 sensor.

  BDMA_ch0 uses Req_Gen0 to generate BDMA requests. The generation of BDMA requests is triggered by the rising edge of lptim2_out signal. This signal is generated by LPTIM2 each 1 s.

- **BDMA_channel1 (BDMA_ch1)** is used to transfer data from SRAM4 to I2C4_TXDR register, in order to send a specified address to HTS221 sensor.

  BDMA_ch1 request is generated when I2C4 TX-FIFO is not full (enable TXDMAEN bit in I2C4_CR1 register).

  The DMAMUX SYNC1 block is programmed to generate a pulse on its dmamux2_evt1 output when the BDMA request is complete.

- **BDMA_channel2 (BDMA_ch2)** is used to transfer data from SRAM4 to I2C4_CR2 register, in order to configure I2C4 interface to request read transfer from HTS221 sensor.

  BDMA_ch2 uses Req_Gen2 to generate BDMA requests. The generation of BDMA requests is triggered by the falling edge of dmamux2_evt1.

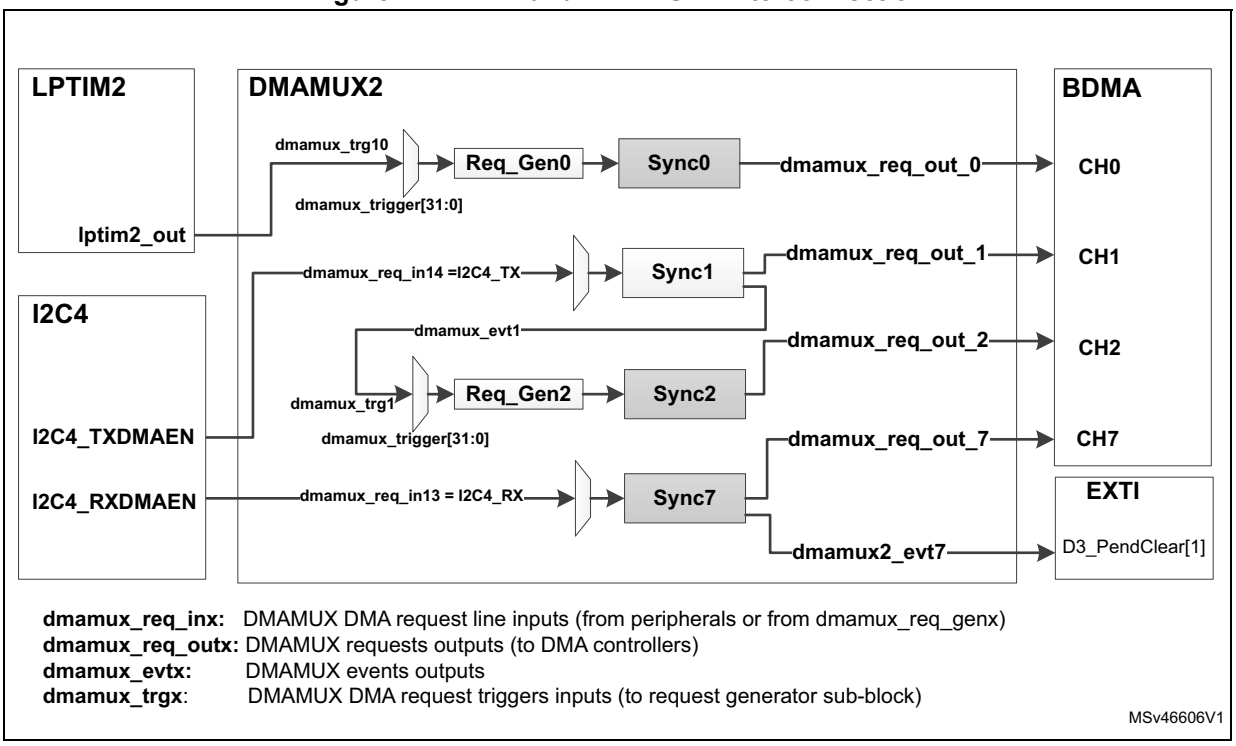
- **BDMA_channel7 (BDMA_ch7**) is used to transfer the received data from I2C4_RXDR register (data sent from HTS221 sensor) to SRAM4. BDMA_ch7 transmission is enabled by setting RXDMAEN bit.

  BDMA_ch7 is configured to generate interrupts to wake up the CPU from CStop mode each time after half-transfer complete (10 bytes of data) or after full transfer complete (20 bytes of data).

  The DMAMUX2 SYNC7 block is programmed to generate a pulse on its dmamux2_evt7 output when the BDMA request is complete. The dmamux2_evt7 signal is used by the EXTI to switch back the D3 domain to Stop mode.

The figure below shows the active signal paths via DMAMUX2.

Figure 11. BDMA and DMAMUX2 interconnection



LPTIM2 programming

LPTIM2 is programmed to still operate when the D3 domain enters Stop mode since it uses ck_lsi clock.

LPTIM2 is programmed to wake up the D3 domain from Stop mode each 1 s and to start BDMA_ch0 transfer through Req_Gen0.

I2C programming

I2C4 is configured to use ck_hsi as kernel clock and programmed to generate BDMA request when TX-FIFO/RX-FIFO is Empty/Full.

A dedicated function called `I2C4_ENABLE_DMA_REQUEST()` is defined in common.h file to enable BDMA requests generation by the I2C4.

8.2.3 Overall description of Mode 4 example

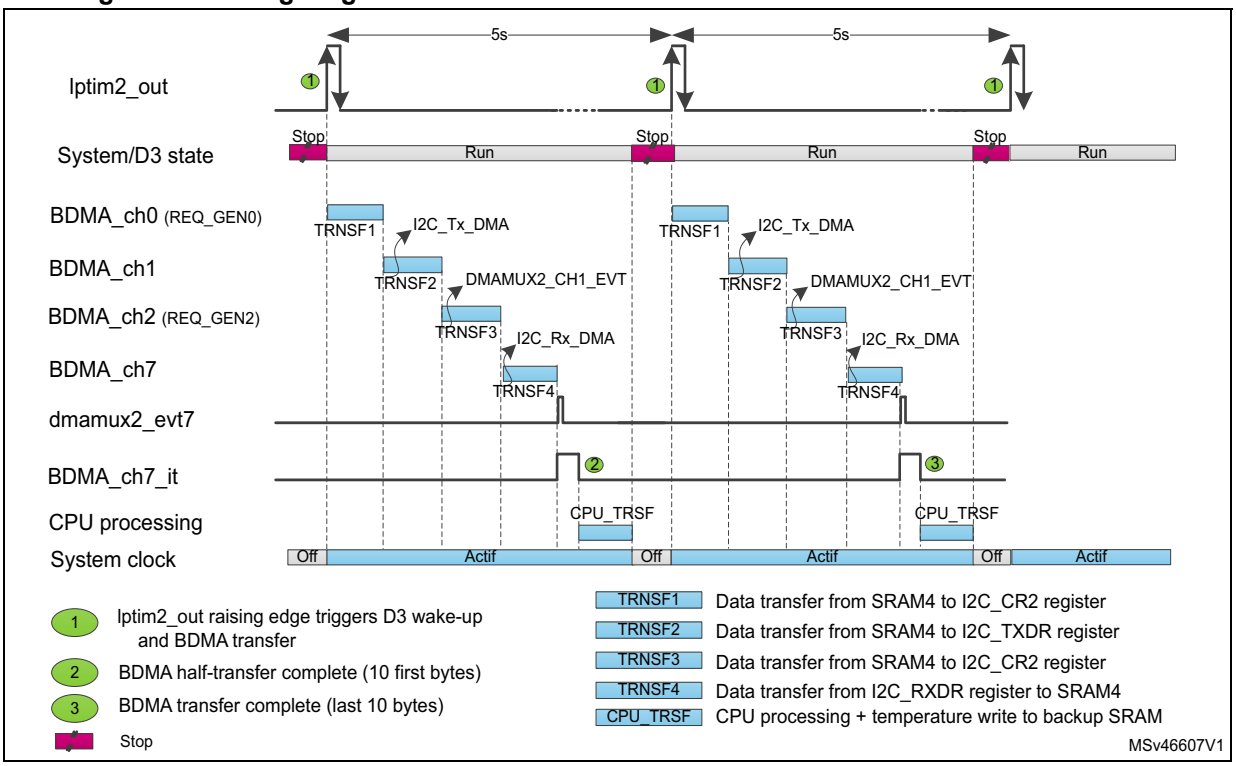
Figure 12 illustrates the timing diagram of I2C4 transmission with D3 domain in Autonomous mode. Detailed steps are the following ones:

- The lptim2_out signal wakes up the D3 domain from Stop mode each 1 s.
- When the D3 domain wakes up from Stop mode, the rising edge of lptim2_out signal triggers BDMA_ch0 to begin data transfer (TRNSF1).
- As soon as I2C4 TX-FIFO is empty, BDMA_ch1 begins data transfer from SRAM4 to I2C4_TXDR register (TRNSF2).
- The end of this transfer triggers BDMA_ch2 transfer (through dmamux2_ch1_evt) from SRAM4 to I2C_CR2 register (TRNSF3).
- As soon as I2C4 RX-FIFO is full, BDMA_ch7 transfers 2 bytes of data from I2C4_RXDR register to SRAM4 (TRNSF4).
- The end of this transfer triggers a dmamux2_evt7 signal which is used to clear the D3_PendClear bit.
- The whole system returns to Stop mode.
- After 5 s and when the expected amount of data has been transmitted (NDT bits of BDMA_CNDTR0 set to 10 indicating half-transfer is complete), the BDMA_ch7 generates an interrupt to wake up the CPU from CStop mode.
- The CPU processes the data located in SRAM4 to be transferred, and copies them to BackupSRAM (CPU_TRSF). The data stored in BackupSRAM are retained while the system is in Stop mode.
- The CPU returns to CStop mode and the whole system return to Stop mode.

LPTIM2 continues to wake up the D3 domain after each 1 s, and BDMA channels continue to do all transfers as mentioned above.

When the expected amount of data has been transmitted (NDT bits of BDMA_CNDTR0 set to 20 indicating transfer is complete), the BDMA_ch7 generates an interrupt to wake up the CPU from CStop mode. The CPU processes the data located in SRAM4 to be transferred, and copies them to BackupSRAM. (CPU_TRSF).

Figure 12. Timing diagram of I2C4 transmission with D3 domain in Autonomous mode



8.3 How to use the application

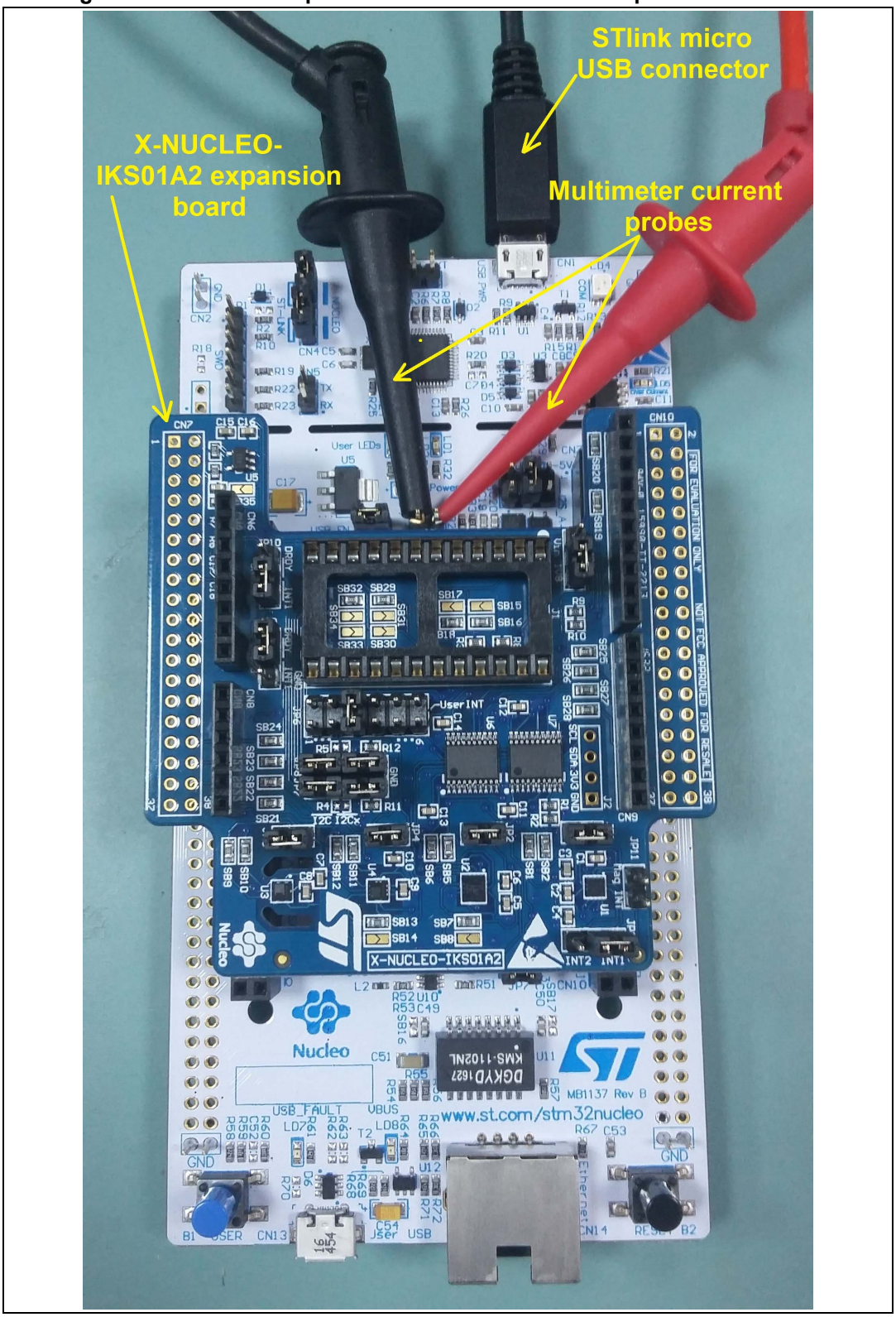
8.3.1 Hardware requirements

To setup the application, the user needs:

- the STM32H743ZI-Nucleo board
- the X-NUCLEO-IKS01A2 expansion board to measure temperature values from HTS221 sensor via I^2C bus

  The X-NUCLEO-IKS01A2 expansion board must be plugged on the matching pins of the STM32 Nucleo board connector.
- to connect the board to a computer through ST-Link with a mini USB for programming and debugging
- a multimeter to measure the current consumption

Figure 13. Hardware requirements and current consumption measurement



1. Picture is not contractual.

8.3.2 Software requirements

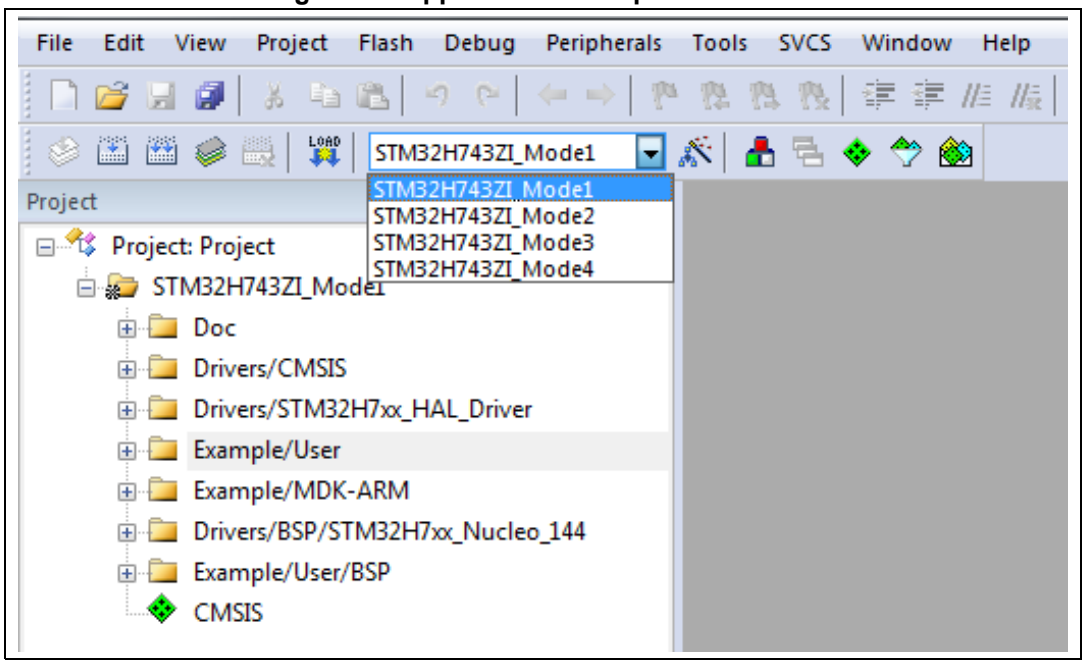
This example is provided with four workspaces described in the table below.

Table 17. Workspaces available in the example

Workspace	Conditions of temperature acquisition			
	CPU	D1 domain	D2 domain	D3 domain
STM32H743ZI_Mode1	CSleep	DRun	DRun	DRun
STM32H743ZI_Mode2	CStop	DRun	DStandby	DRun
STM32H743ZI_Mode3	CStop	DStandby	DStandby	Run
STM32H743ZI_Mode4	CStop	DStandby	DStandby	Run / Stop

The user needs to choose the dedicated workspace to execute the selected mode.

Figure 14. Application workspace selection



8.3.3 Running the example

The figures below show the different messages displayed on the terminal after running each mode.

Figure 15. Message after running Mode 1

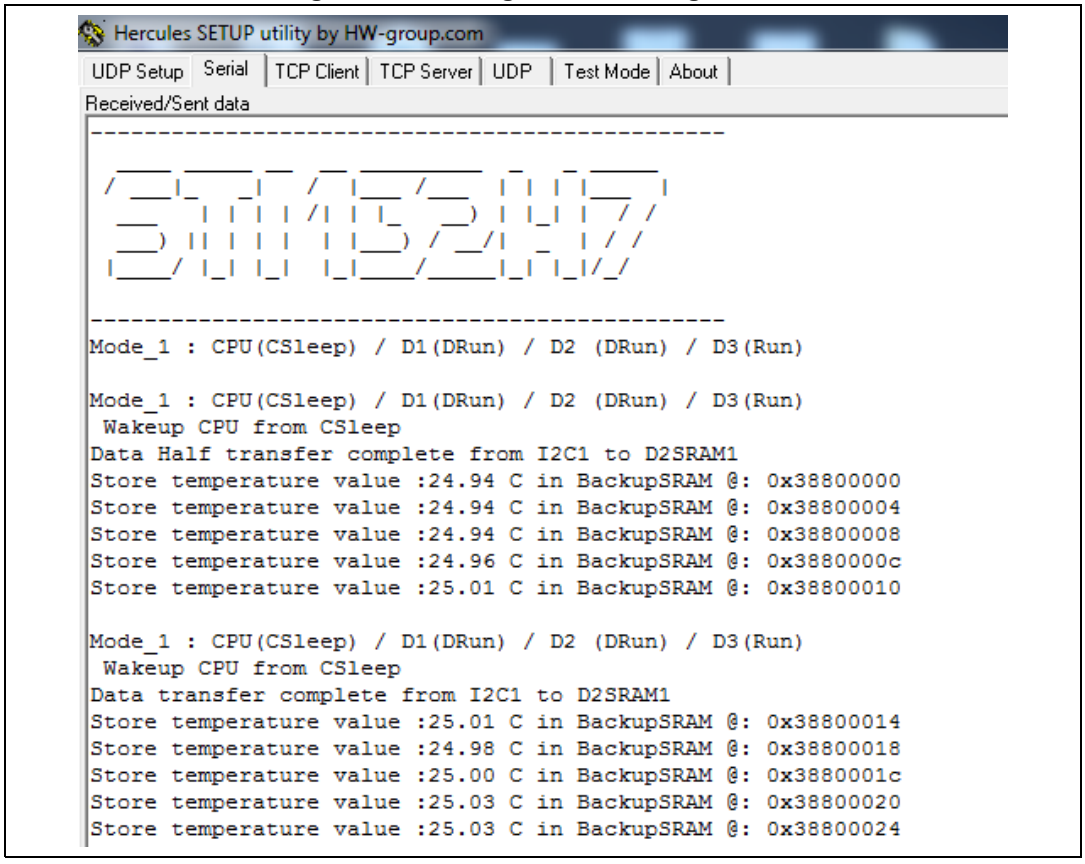


Figure 16. Message after running Mode 2

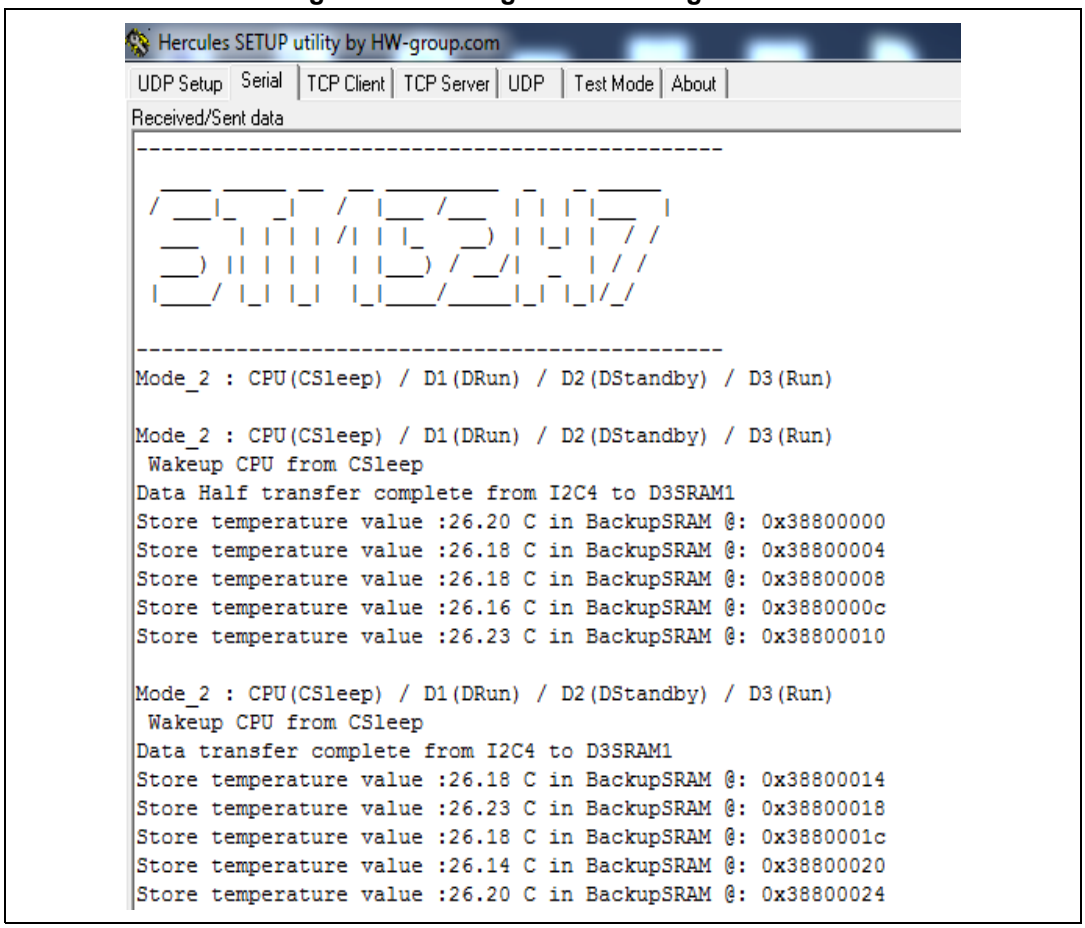


Figure 17. Message after running Mode 3

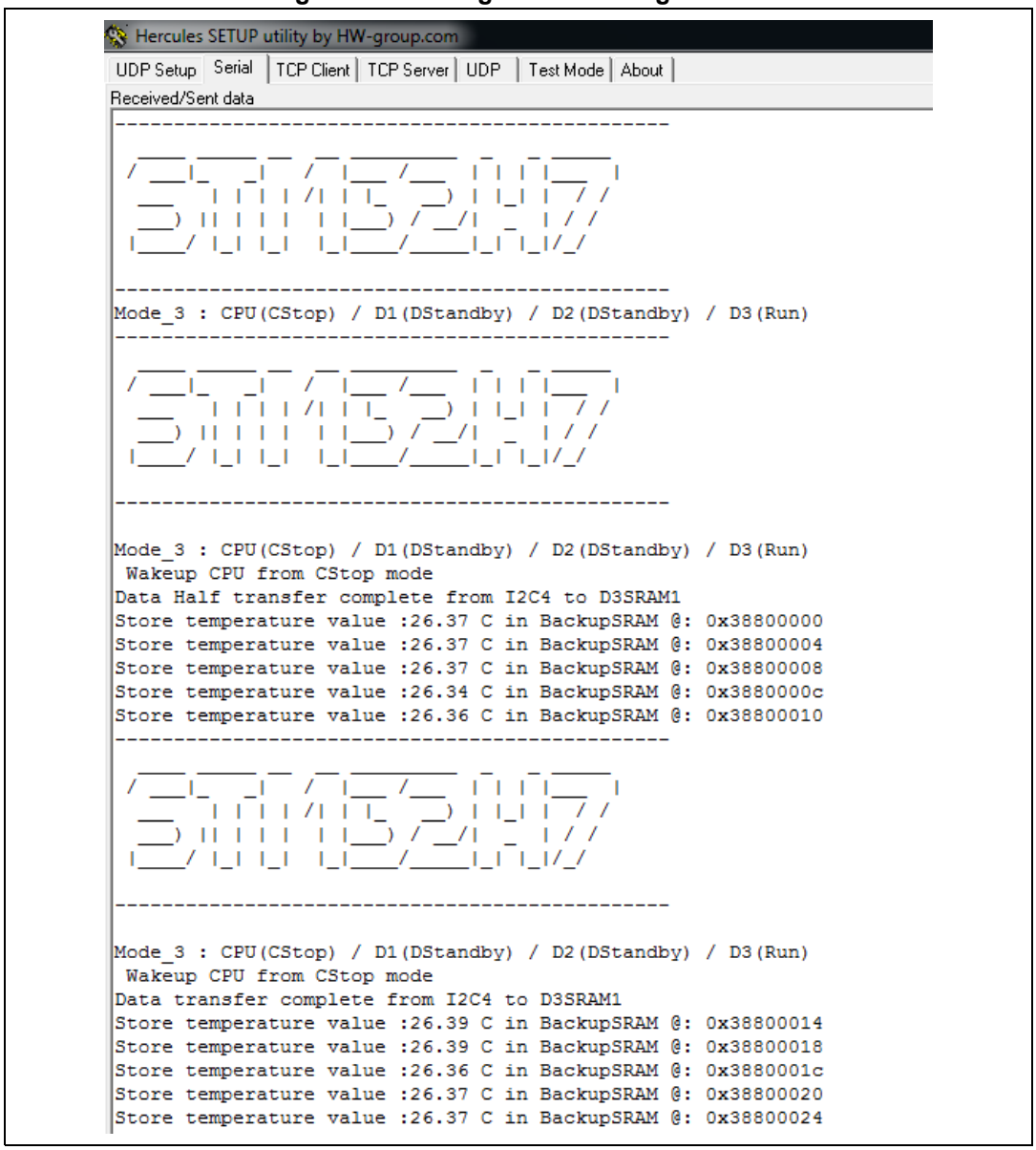
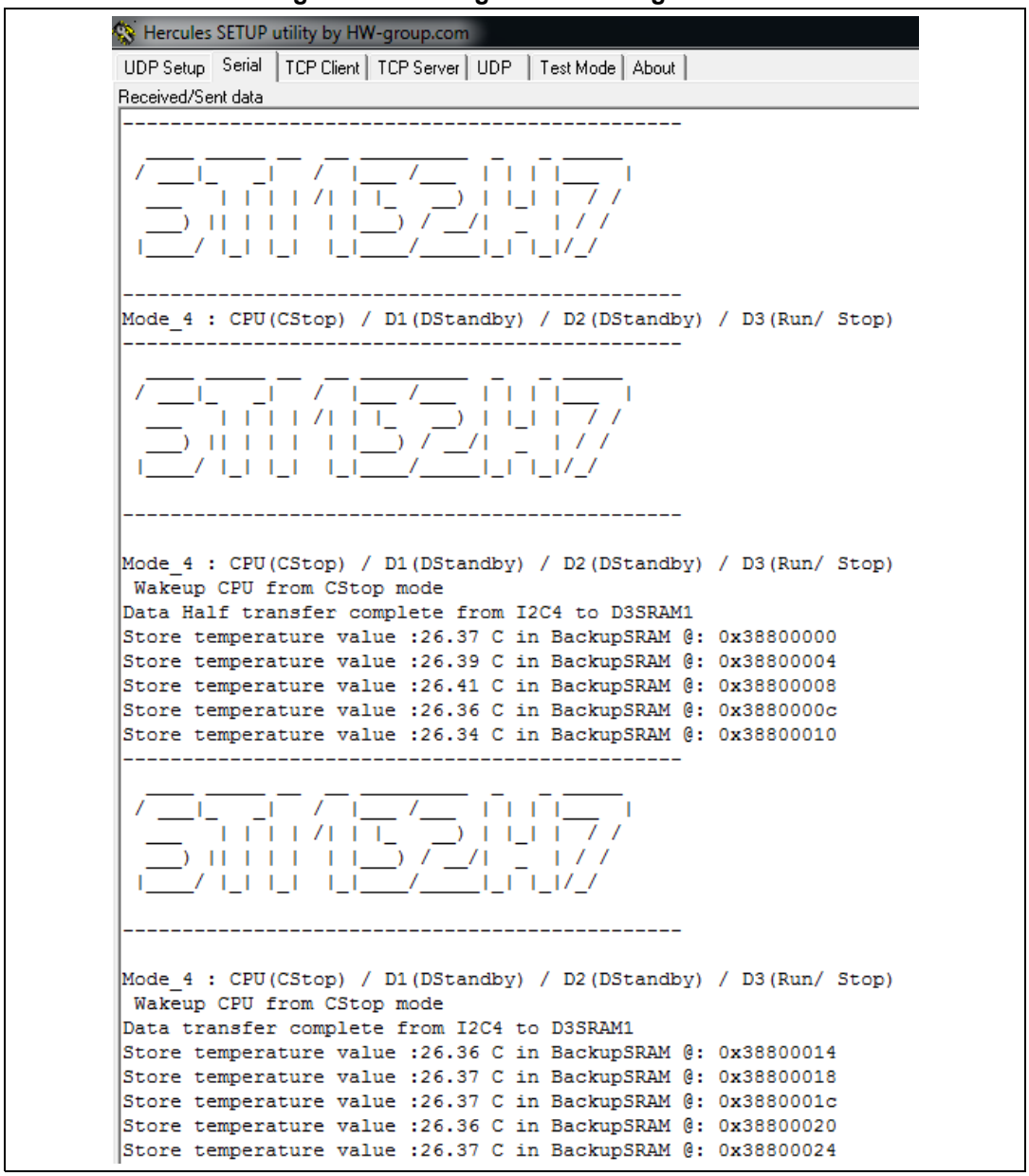


Figure 18. Message after running Mode 4



8.4 Power consumption measurements

The measurements below have been done with the following settings:

- voltage = 3.3 V
- system clock HCLK = 64 MHz

The table below describes the measured current consumption in all modes.

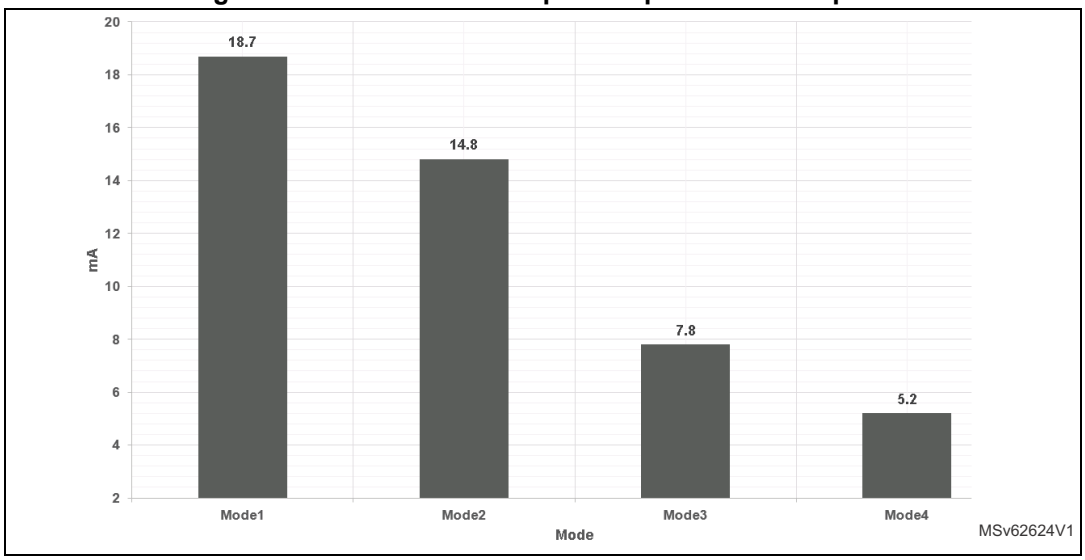
Table 18. MCU power consumption in different modes

Mode	CPU state	D1 domain	D2 domain	D3 domain	Average MCU current consumption (mA)
Mode1	CSleep	DRun	DRun	Run	18.7
Mode2	CSleep	DRun	DStandby	Run	14.8
Mode3	CStop	DStandby	DStandby	Run	7.8
Mode4	CStop	DStandby	DStandby	Run/Stop	5.2

Synthesis

The figure below summarizes the STM32H7x3 devices state domains effects on current consumption optimization.

Figure 19. Domains state impact on power consumption



This figure illustrates the benefits of STM32H7x3 devices smart architecture that manages power supply per domain. Up to 72 % (Mode4) of power consumption is saved compared with an application running only with basic low-power optimization (using Sleep mode only: Mode1).

The new feature, allowing one sub-domain to be disabled while the two other domains remain in Run, enhances the power consumption in our example by 21 % (Mode2).

Activating the batch acquisition mode (D3 domain in Autonomous mode) decreases Mode2 power values down to 7,8 mA and save up to 47 % of power consumption.

Enhancing D3 Autonomous mode by allowing D3 to enter Stop mode when there is no communication ongoing (mode4), saves more power consumption compared to mode3 (up to 33 %).

More power efficient implementations are also available. As an example, the system clock frequency ca be reduced by using the low-power internal oscillator (CSI at 4 MHz) as wakeup clock from Stop mode. This implementation is possible when using D3 Autonomous mode (in Mode 4: switching from Run to Stop), since CSI can be selected as wakeup clock from system Stop mode (STOPWUCK bit in RCC_CFGR register).

9 Revision history

Table 19. Document revision history

Date	Revision	Changes
9-Jun-2017	1	Initial release.
15-May-2019	2	Updated: – *Section 1: System architecture* – *Section 2.1: Voltage regulator (LDO) supply* – *Section 2.2: Voltage regulator bypass* – *Figure 5: Power control modes detailed state diagram*