STM32MP15x lines using low-power modes

Introduction

STM32MP15x lines devices are built on an Arm® Cortex®-A7 with single or dual-core MPU subsystem combined with an Arm® Cortex®-M4 CPU.

STM32MP15x devices can be configured in various low-power modes in order to reduce power consumption when necessary.

This application note explains the various low-power modes of the STM32MP15x lines devices, and how to configure and exit from them. This document also presents some guidance to be considered when using low-power modes at system level, and when using an external STPMIC1x power regulator component.

Table 1. Applicable products

Type	Product lines
STM32MP15x	STM32MP151, STM32MP153, STM32MP157

In this document, *"STM32MP15x"*, refers specifically to STM32MP151, STM32MP153 and STM32MP157 lines devices.

For further information on STM32MP15x lines devices, refer to the following documents and deliverables available on www.st.com.

- STM32MP1 Series reference manuals (see Table 2. STM32MP15x lines configuration for details)
- STM32MP1 Series datasheets
- STPMIC1x datasheet
- *Getting started with STM32MP1 Series hardware development* (AN5031)
- *STM32MP1 Series and STPMIC1x hardware and software integration* (AN5089)
- STM32CubeMP1 package
- STM32MP1 Series embedded software

1 Overview

This application note is applicable to all the devices of the STM32MP15x lines. The table below describes their main characteristics.

Depending on the device's part number, the system includes a Cortex-M4 and either a single-core or a dual-core Cortex-A7. In this document, the Cortex-A7 is called MPU and the Cortex-M4 is called MCU.

The full featured system (see the table below) is partitioned in:

- One MPU subsystem: dual Cortex-A7 with L2 cache
- One MCU subsystem: Cortex-M4 with associated peripherals clocked according to CPU activity

The present document assumes a full featured device (for example STM32MP157).

Table 2. STM32MP15x lines configuration

Lines	Reference manual	Cortex-A7 configuration	Cortex-M4	GPU	DSI	FDCAN
STM32MP151	RM0441	Single-core	Yes	No	No	No
STM32MP153	RM0442	Dual-core	Yes	No	No	Yes
STM32MP157	RM0436	Dual-core	Yes	Yes	Yes	Yes

In this document MCU subsystems are sometimes referred to as "MCU" and MPU subsystems as "MPU" to facilitate the reading of the document.

This document applies to Arm®-based devices.

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

2 Glossary

Table 3. **Glossary**

Term	Meaning
AHB	Advanced high-performance bus
ATF	Arm® trusted firmware
AVD	Analog voltage detector
BKPSRAM	Backup SRAM
BOR	Brownout reset
BUCK	Step down switched-mode power-supply converter
CSS	Clock security system
DDR	Double data-rate (SRAM)
DDRCTRL	DDR controller
DDRPHYC	DDR physical interface control
DSI	Display serial interface
ETH	Ethernet controller
FDCAN	Controller area network with flexible data-rate. Could also support time triggered CAN (TT)
GPIO	General purpose input output
GPU	Graphic processing unit
HAL	Hardware abstraction layer
HDP	Hardware debug port
IRQ	Interrupt request
IWDG	Independent watchdog
LDO	Low dropout regulator
LpDDR	Low power DDR
LSE	Low-speed external quartz oscillator
LSI	Low-speed internal oscillator
MCU	Microcontroller
MLAHB	Multi layer AHB / AHB based interconnect
MPU	Microprocessor
PLL	Phase locked loop
PVD	Programmable voltage detector
PWR	Power control block
QSPI	Quad data lanes serial peripheral interface
RCC	Reset and clock control
RETRAM	Retention memory
RTC	Real time clock
SDMMC	Secure digital and MultiMedia card interface. Supports SD, MMC, eMMC and SDIO protocols
SMP	Symmetric multiprocessing
SPL	Secondary program loader

Term	Meaning
SRAM	Static random access memory
STPMIC1x	Power management integrated circuit. External circuit that provides various platform power supplies with large controllability through signals and serial interface
SYSRAM	System SRAM
SW	Software
DTS	Digital Temperature Sensor
TEMPH-L	Temperature sensor high-low monitoring
USART	Universal synchronous asynchronous receiver transmitter
USB OTG	Universal Serial Bus (USB) on-the-go (OTG). A standard USB interface able to become host or device
VBATH-L	VBAT high-low monitoring
WFE	Wait for event
WFI	Wait for interrupt

3 Power management concept

This section describes the STM32MP15x lines high-level power management concept.

Refer to the corresponding reference manual for more details (see Table 2. STM32MP15x lines configuration).

3.1 STM32MP15x lines system architecture

The STM32MP15x devices are based on a main dual-core MPU subsystem and a MCU subsystem. The multiple core architecture requires specific power modes both at system and at individual MPU and MCU subsystem level.

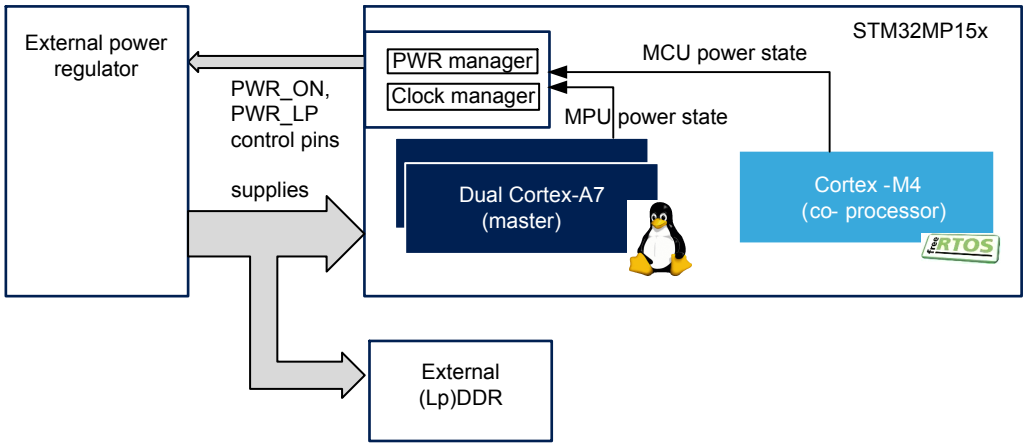
Internal digital logic is supplied externally in STM32MP15x lines devices, this is different than most STM32 MCUs which have an internal LDO (low dropout regulator) for such purpose. This key difference results in a different power-supply management for the STM32MP15x devices which require the use of external signals and/or programming of the external regulator to control the desired voltage supplies.

The power management features are spread between the RCC (reset and clock control) and the PWR (power) blocks of the STM32MP15x lines device.

- The RCC block ensures the clock tree handling (PLLs, muxes, dividers, clock gating) and the resets (local resets of the peripherals, Cortex-A7 reset, Cortex-M4 reset, platform reset)
- The RCC block permits to select the power mode based on the respective power states of the MCU and MPU subsystems
- The PWR block is responsible for low-power entry/exit. It drives the control pins (PWR_ON, PWR_LP) to the external regulator based on the power mode.

The figure below shows the STM32MP15x lines high-level system architecture.

Figure 1. STM32MP15x lines high-level system architecture



3.2 System supplies (V_{DD} and V_{DDCORE})

The STM32MP15x lines devices require many power supplies that are detailed in the corresponding datasheet and reference manual (see Table 2. STM32MP15x lines configuration). Among those power supplies, V_{DD} and V_{DDCORE} play a key role in the low-power mode configuration.

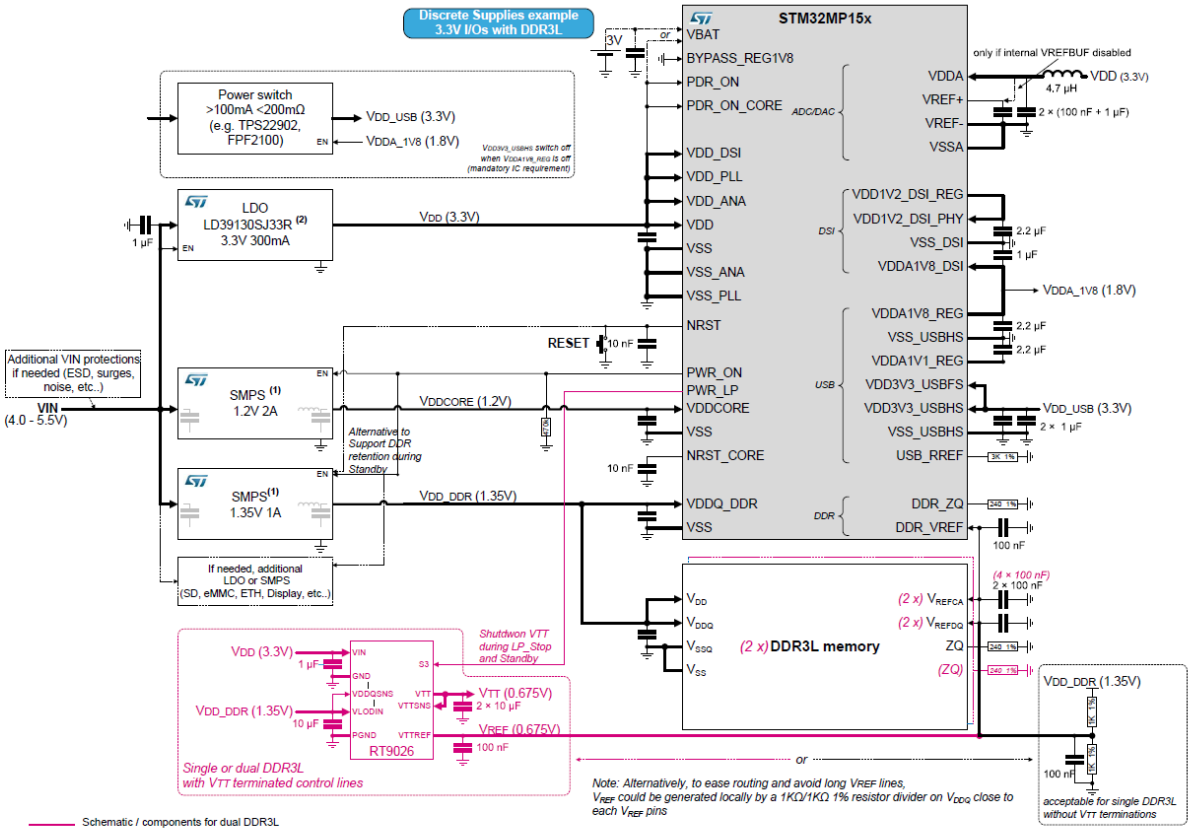
- V_{DD} power supply input for IOs and system analog such as reset, power management oscillators and PLLs.
- V_{DDCORE} digital core domain supply, dependent on V_{DD} supply. V_{DD} shall be present before V_{DDCORE}.

The various power pins of the STM32MP15x devices can be supplied either by some external discrete supplies or by using STPMIC1x as detailed in the reference design section of the application note *Getting started with STM32MP1 Series hardware development* (AN5031).

The STPMIC1x has several part numbers. Find below the part numbers appropriate for the STM32MP15x devices:

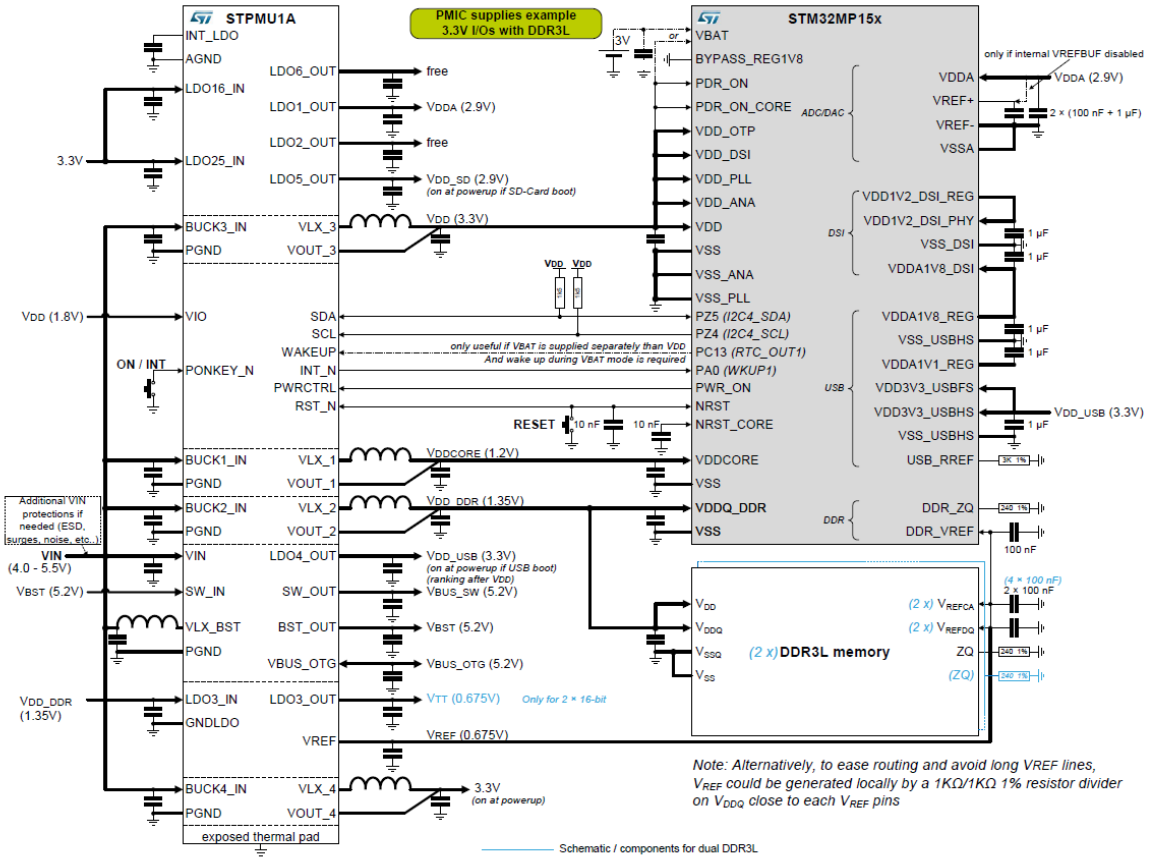
- STPMIC1APQR for application using VDD = 3.3 V
- STPMIC1BPQR for application using VDD = 1.8 V

Figure 2. Discrete supplies example 3.3V I/Os with DDR3L



Note: Several types of DDR are supported (LpDDR2, LpDDR3, DDR3), this impacts the low-power mode management. Refer to Figure 7. Available power state transitions.

Figure 3. STPMIC1x example 3.3 V I/Os with DDR3L



The STPMIC1x provides the power distribution. It is driven by I²C and by control signals coming from the STM32MP15x device. Control is done by the Cortex-A7 MPU subsystem core. The main benefit of using the STPMIC1x versus using discrete regulator supplies is to have an integrated, smaller and cheaper solution. Refer to application note *STM32MP15x and STPMIC1x HW and SW integration* (AN5089) for more details.

4 Operating modes

This section describes the STM32MP15x lines various power modes and the ways of activating them.

4.1 Operating modes description

The operating modes allow the control of the clock distribution to the different system parts and the control of the power supplies of the system. The system operation mode is driven by the MPU subsystem and the MCU subsystem.

The power management controls the V_{DDCORE} supply in accordance with the system operating modes.

The table below presents the operating modes available for the system and for the MPU and MCU subsystems. The power mode at system level is dependent on the power modes at each subsystem level (MPU and MCU).

Table 4. Power modes

-	Power mode	Description	Notes
System	Run	V_{DDCORE} power on, clock on	-
	Stop	V_{DDCORE} power on, clock off	-
	LP-Stop	V_{DDCORE} power on, clock off	(1)
	LPLV-Stop	V_{DDCORE} reduced power level and supply load, clock off	(1)(2)
	Standby	V_{DDCORE} power off, clock off	-
	VBAT	V_{SW} is supplied by battery, V_{DDCORE} is off, RTC is still active clocked by LSE crystal	(3)
	Power off	All power supplies off	-
MPU / MCU	CRun	V_{DDCORE} power on, clock on	-
	CSleep	V_{DDCORE} power on, CPU clock off, peripheral clock on/off	(4)
	CStop	V_{DDCORE} power on, CPU subsystem clock off	-
MPU	CStandby	V_{DDCORE} power on or off, clock off	(5)

1. *There is no difference in the output control pins PWR_ON and PWR_LP between LP-Stop and LPLVStop (see Section 4.3 External control signals PWR_ON, PWR_LP pins). While both LP-Stop and LPLV-Stop can be activated using STPMIC1x through programming of its internal register prior entering the targeted low-power mode (refer to Section 4.3.1 Using STPMIC1x power regulator), both modes may not be available using other external power regulators.*
2. *The main difference with Stop mode is that PWR_ON output signal is toggling to 0 when using STPMIC1x external power regulator, enabling the possibility to take actions like powering off DDR termination resistance power supply.*
3. *To retain the content of the VSW domain (RTC, backup registers, backup RAM and retention RAM) when V_{DD} is turned off, the VBAT pin can be connected to an optional standby voltage supplied from a battery or from another source.*
4. *The MCU/MPU subsystems allocated peripheral(s) clock operate according to RCC PERxLPEN. When the MPU enters CSleep with WFE (wait for event), the MPU subsystem allocated peripheral(s) clock operate as in MPU CRun mode, irrespective of RCC PERxLPEN.*
5. *In CStandby mode, the MPU is in power off only if the system mode is in Standby (when MCU is in CStop with PDDS=1). Refer to Table 6. Power-system mode versus sub-system mode summary.*

The choice of which low-power mode to use is dependent on power saving target and it must be chosen according to which wakeup interrupts are needed/available in the chosen low-power mode and the expected wakeup time duration.

User should always make sure that the chosen low-power mode is consistent with the available wakeup sources. For instance LPLV-Stop or Standby modes have very limited wakeup source capabilities.

Also the wakeup duration is longer if power supplies voltage have been reduced or switched off by reducing V_{DDCORE}, switching off DDR resistance termination supply with DDR in self refresh or switching off the full DDR (which may require reloading the firmware from the Flash memory). The system's low-power mode wakeup capabilities are presented in the below table.

Table 5. **System's low-power mode wakeup capabilities**

System power mode	Wakeup sources
Stop/LP-Stop	DBG, PVD, AVD, USBH, OTG, CEC, ETH, MDIOS, USARTx, I2Cx, SPIx DTS, LPTIMx, GPIOs
LPLV-Stop	PVD, AVD, DTS, LPTIMx, GPIOs
Standby	Six GPIO pins: PA0, PA2, PC1, PC13, PI8, PI11
All modes	BOR, VBATH/VBATL, TEMPH/TEMPL, LSE CSS, RTC/auto wakeup, tamper pins, IWDGx

4.2 Low-power mode control

4.2.1 Low-power mode control registers

The control register bits presented below are related to the low-power modes (for description of low-power modes see Section 4 Operating modes).

PDDS bit in register PWR MPU control register (PWR_MPUCR)

0: Keeps Stop mode when MPU enters to CStop.

1: Allows Standby mode when MPU enters to CStop.

When CSTBYDIS = 0 allows MPU to enter CStandby mode.

PDDS bit in register PWR MCU control register (PWR_MCUCR)

0: Keeps Stop mode when MCU enters to CStop.

1: Allows Standby mode when MCU enters to CStop.

CSTBYDIS bit in register PWR MPU control register (PWR_MPUCR)

0: MPU CStandby mode enabled.

1: MPU CStandby mode disabled.

Both MPU and MCU can access the PWR_MPUCR and PWR_MCUCR. However the PWR_MPUCR can be set in secure mode (Linux case) to only allow MPU write access.

MCU can write the PWR_MCUCR register bits using HAL_PWR_EnterStandbyMode() function.

- The HAL function HAL_PWR_EnterStandbyMode() sets the MCU PDDS bit to '1' and enters the CStop mode for the MCU subsystem through a WFI (wait for interrupt). The Standby mode is entered only once the MPU subsystem enters the CStop mode with PDDS=1 & CSTBYDIS=1 or the CStandby mode (see Section 4.4: Low-power mode entry sequence)

The tables below detail the possible combinations of the power modes between system and MPU/MCU subsystems and the system's low-power modes summary respectively.

Table 6. Power-system mode versus sub-system mode summary

-		MPU[1]				
		CRun	**CSleep**	**CStop (PDDS = 0)**	**CStop (PDDS = 1 CSTBYDIS=1)**	**CStandby[2]**
MCU	**CRun**	Run	Run	Run	Run	Run[3]
	CSleep	Run	Run	Run	Run	Run[3]
	CStop (PDDS = 0)	Run	Run	Stop LP-Stop LPLV-Stop	Stop LP-Stop LPLV-Stop	Stop[3] LP-Stop LPLV-Stop
	CStop (PDDS = 1)	Run	Run	Stop LP-Stop LPLV-Stop	Standby[4]	Standby[4]

1. *Refer to Section 4.6.7 Exit from MPU CStop and CStandby for additional details on CStop and CStandby.*
2. *CStandby mode on MPU is not supported in the STM32MPU OpenSTLinux Distributions. System standby is reached with MPU in CStop mode (PDDS=1, CSTBYDIS=1).*
3. *When exit from MPU CStandby with System in Run, Stop, LP-Stop or LPLV-Stop, program execution restarts with a local MPU reset. This 'reboot' can affect MCU operation due to possible re-initialization of some peripheral registers by the BootROM when reloading MPU initialization code from external Flash.*
4. *When exit from MPU CStandby or CStop (PDDS=1, CSTBYDIS=1) with System in Standby, program execution restarts in the same way as after a power on reset (option bytes loading, reset vector fetched, or other).*

Table 7. System low-power modes summary

System	MPU	DDR[1]	MCU	Sys-oscillators[2]	hclk4	PWR_LP	PWR_ON	
							LPCFG = 0	LPCFG = 1
Run	CRun or Csleep	Active / Auto refresh	CRun or CSleep	On	On	1	1	1
	CStop or CStandby	Self refresh						
	CRun or CSleep	Active / Auto refresh	CStop					
Stop (LPDS = 0)[3]	CStop or CStandby	Self refresh	CStop	On/Off[4]	Off			
LP-Stop LPLV-Stop[3] (LPDS = 1)						0[5]		0[5]
Standby	CStandby or (CStop and MPU PDDS = 1 and MPU CSTBYDIS = 1)	Off / Self refresh	CStop and MCU PDDS = 1	Off		0[6]	0[6]	0[6]

1. *The DDR operation state is only presented for information. For example, the DDR may be Off when the MPU is in CRun like when executing from BootROM.*
2. *The sys-oscillators are the HSI, HSE or CSI oscillators configured ON in RCC.*
3. *At least 1 PDDS bit (MCU or MPU) selects Stop (PDDS = 0).*

4. *When the system oscillator HSI, HSE or CSI is used, the state is controlled by the respective xxxKERON (HSIKERON, HSEKERON or CSIKERON bits present in the RCC oscillator clock enable set register (RCC_OCENSETR) and the RCC oscillator clock enable clear register (RCC_OCENCLRR)), else the system oscillator is off.*
5. *Settings in PWR control register 3 (PWR_CR3) bits POPL have no impact on the LP-Stop and LPLV-Stop mode PWR_ON and PWR_LP pulse low time.*
6. *A guaranteed minimum PWR_ON and PWR_LP pulse low time can be defined in PWR control register 3 (PWR_CR3) bits POPL.*

4.3 External control signals PWR_ON, PWR_LP pins

The two output pins PWR_ON and PWR_LP are related to the VDDCORE supply. They are used by external component/regulator to identify which voltage level should be applied on V_{DDCORE}. Refer to Section 4 Operating modes for more details on PWR_ON, PWR_LP configuration.

- PWR_ON: V_{DDCORE} supply request

  It is automatically generated by the hardware depending on the state of the STM32MP15x device and on the value of **LPCFG** (PWR_ON pin configuration) bit in PWR control register 1 (PWR_CR1).
- PWR_LP: V_{DDCORE} low-power mode control (active low)

  It is automatically generated by the hardware depending on the state of the STM32MP15x device and the value of **LPDS** (low voltage deepsleep LPLV-Stop mode selection) bit on PWR control register 1 (PWR_CR1).

The table below indicates the PWR_ON, PWR_LP output pin values depending on the various configuration of power modes and LPDS, LPCFG, LVDS bits.

- LPDS is PWR control register 1 (PWR_CR1) bit 0
- LPCFG is PWR control register 1 (PWR_CR1) bit 1
- LVDS is PWR control register 1 (PWR_CR1) bit 2

The table also shows the differences in the usage of pins PWR_ON and PWR_LP depending if the STM32MP15x device is to be used with STPMIC1x or with other external power supply components.

The power modes are detailed in the PWR section of the corresponding reference manual (see Table 2. STM32MP15x lines configuration).

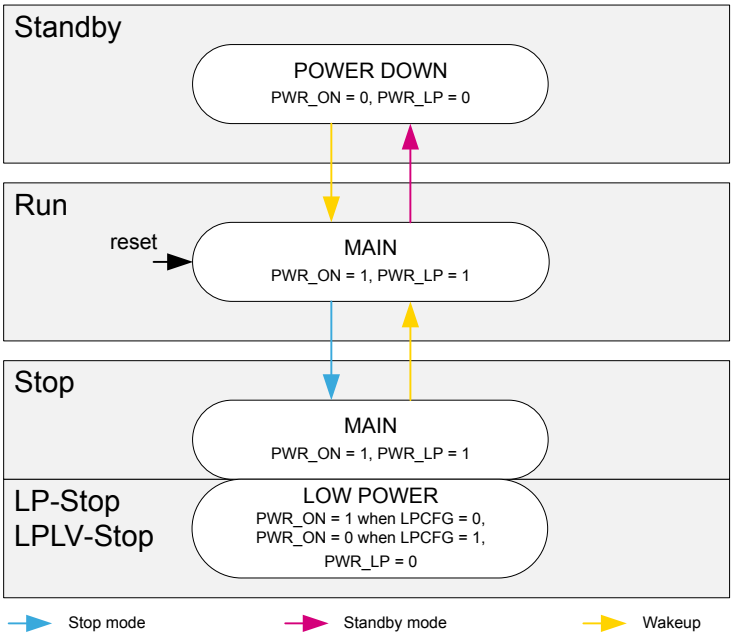
Table 8. PRW_ON, PWR_LP levels according power modes, LPDS, LVDS and LPCFG bits

STM32MP15x state	LPDS/LVDS bits	LPCFG bit	PWR_ON	PWR_LP	V_{DDCORE}
Startup (until V_{DD} reaches POR threshold level)	X/X	X	0	0	Off
Run mode	X/X	X	1	1	On
Stop mode	0/X	X	1	1	On
LP-Stop mode	1/0	0	1	0	On
		1(1)	0(2)	0(2)	
LPLV-Stop mode	1/1	0(3)	1	0	On(4)
		1(1)	0(2)	0(2)	
Standby mode	0/0	X	0(2)	0(2)	Off
VBAT mode (V_{DD} powered down)	X/X	X	HiZ	HiZ	Off

1. *Configuration used with STPMIC1x PWRCTRL pin connected to PWR_ON.*
2. *There is no difference between LP-Stop, LPLV-Stop and Standby mode on the PWR_ON, PWR_LP output values '00' with LPCFG bit=1.*
3. *Depending on the external power regulator that is used, this configuration might not be available (not possible to reduce the power supply level).*
4. *V_{DDCORE} supply level and power load can be decreased.*

The figure below illustrates the relation between the system states and the PWR_ON, PWR_LP output pins for regulator control for STM32MP15x lines devices.

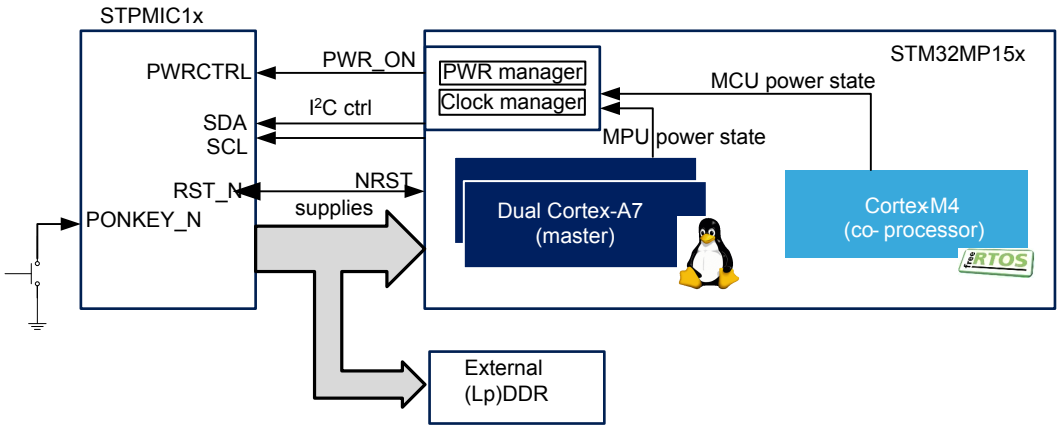
Figure 4. Power states and external regulator control



4.3.1 Using STPMIC1x power regulator

The figure below shows the main differences at system level when using the STPMIC1x power regulator.

Figure 5. STM32MP15x lines high-level system architecture when using STPMIC1x



When using STPMIC1x power regulator, only PWR_ON output control pin from STM32MP15x is needed for the control of the STPMIC1x power modes (through PWRCTRL pin). In that case, the user shall set the LPCFG bit of the PWR_CR1 register to '1'.

Table 8. PRW_ON, PWR_LP levels according power modes, LPDS, LVDS and LPCFG bits highlights on the gray cells that there is no difference between LP-Stop, LPLV-Stop and Standby mode on the PWR_ON, PWR_LP output values '00' with LPCFG bit=1.

The STPMIC1x differentiates between those LP-Stop, LPLV-Stop and Standby low-power modes and apply the correct V_{DDCORE} value thanks to STPMIC1x internal registers. Those registers programming is done via the I2C interface and not through the PWR_ON, PWR_LP pins values.

When using STM32MPU OpenSTLinux Distributions, this programming is handled by the first stage bootloader (TF-A or OPTEE). Hence before entering LP-Stop, LPLV-Stop or Standby mode the application shall configure the STPMIC1x internal registers to the desired system power supplies level (like V_{DDCORE}, V_{DD_DDR}, and others).

When entering LP-Stop mode, despite the fact that V_{DDCORE} level is not decreased (only in LPLV-Stop), it is still possible to program the STPMIC1x so that other power supplies are shut down if not needed (for example the power supply of the DDR termination resistances).

The STPMIC1x includes two 8-bit registers for each of the BUCK (1,2,3,4) and LDO (1,2,3,4,5,6):

- A BUCKx/LDOx control register (HP mode) that stores the expected output voltage value during Run mode.
- A BUCKx/LDOx control register (LP mode) that stores the expected output voltage during low-power mode.

Each of those registers have the following structure:

7	6	5	4	3	2	1	0
output value <5:0>						hplp	ena

Bits 7:2: **output value**: the output value on 6 bits corresponds to a specified voltage value (refer to STPMIC1x datasheet).

Bit 1: **hplp**: forces high/low load capability (allows more/less load on STPMIC1x output).

0: High load capability

1: Low load capability

Note: hplp bit is applicable only for BUCKx registers, not for LDOx.

Bits 0: **ena**: enable bit.

0: BUCK/LDO disabled

1: BUCK/LDO enabled

The following tables show how the STPMIC1x could be programmed in Run mode and how STPMIC1x could be programmed before entering LP-Stop, LPLV-Stop and Standby modes respectively. Both tables list all power supplies and not only the V_{DDCORE} one and the settings used on them are in line with the example presented on Section 3.2 .

Table 9. STPMIC1x (HP mode) programming for Run mode

Supply name	Control register (HP mode) /@	Run
V_{DDCORE}	BUCK1 / 0x20	0x61 (1.2 V)
V_{DD_DDR}	BUCK2 / 0x21	0x79 (1.35 V)
V_{DD}	BUCK3 / 0x22	0xD9 (3.3 V)
3V3	BUCK4 / 0x23	0xD9 (3.3V)
V_{REF_DDR}	VREFDDR / 0x24	0x1
V_{DDA}	LDO1 / 0x2A	0x51 (2.9 V)
V_{DD_USB}	LDO4 / 0x28	0x1 (3.3 V)
V_{DD_SD}	LDO5 / 0x29	0x51 (2.9 V)
V_{TT}	LDO3 / 0x27	0x7D (0.675 V)(1)

1. *LDO3 output value is BUCK2 output /2 = 0.675 V for DDR3 use case.*

Table 10. STPMIC1x (LP mode) programming for LP-Stop LPLV-Stop and Standby mode

Supply name	Control register (LP mode) /@	LP-Stop	LPLV-Stop	Standby with DDR SR	Standby w/o DDR SR
V_{DDCORE}	BUCK1 / 0x30	0x61 (1.2 V)	0x33 (0.9 V)	0x30 (off)	
V_{DD_DDR}	BUCK2 / 0x31	0x79 (1.35 V)			0x7A (off)
V_{DD}	BUCK3 / 0x32	0xD9 (3.3 V)			
3V3	BUCK4 / 0x33	0xD9 (3.3 V)		0xD8 (off)	
V_{REF_DDR}	VREFDDR / 0x34	0x1			0x0 (off)
V_{DDA}	LDO1 / 0x3A	0x51 (2.9 V)		0x50 (off)	
V_{DD_USB}	LDO4 / 0x38	0x1 (3.3 V)		0x0 (off)	
V_{DD_SD}	LDO5 / 0x39	0x51 (2.9 V)		0x50 (off)	
V_{TT}	LDO3 / 0x37	0x7C (off)			

Note: Setting bit0 to '0' disables the corresponding BUCK/LDO, and its value is set to "off".

Example: V_{DDA} = 2.9 V (0x51) when bit0='1' and is 'off' (0x50) when bit0='0'.

Application note *"STM32MP15x and STPMIC1x HW and SW integration"* (AN5089) provides more details on how to use STPMIC1x.

The NRST pin of the STM32MP15x lines can be activated by the STPMIC1x RST_N pin when the user is setting STPMIC1x PONKEY_N pin to '0', hence resetting both the STPMIC1x and the STM32MP15x devices.

4.3.2 Using other external power supplies

When using external power supplies other than STPMIC1x , the use of both PWR_ON and PWR_LP output control pins is possible in order to control external power supply if low-power modes below Stop mode (such as LP-Stop, LPLV-Stop and Standby) are required by the application.

When the STM32MP15x device goes to LP-Stop or LPLV-Stop mode, PWR_ON shall be set to '1'; when they are on Standby PWR_ON shall be set to '0'. To activate this usage, the user must set the LPCFG bit to '0'.

4.4 Low-power mode entry sequence

The STM32MP15x lines have dedicated power-modes at subsystems level (MPU, MCU) and at system level (see Section 4 Operating modes). This section details how to enter those power modes at subsystem and at system level.

MPU subsystem

The MPU subsystem's low-power modes (CSleep, CStop and CStandby) are entered by the MPU when executing the WFI (wait for interrupt) or WFE (wait for event) instructions. WFE being available only when entering CSleep. In order for the MPU to enter the subsystem's low-power modes, the RCC and PWR registers must be already correctly programmed (refer to Section 5.1 Peripheral assignment and allocation). When using STM32MPU OpenSTLinux Distributions those mechanisms are handled by the first stage bootloader (refer to Section 4.5 Power management under Linux).

MCU subsystem

MCU subsystem's low-power modes (CSleep and CStop) are entered by the MCU when executing the WFI (wait for interrupt), or WFE (wait for event) instructions, or when the SLEEPONEXIT bit in the Cortex®-M4 System Control register is set on Return from ISR.

When using STM32CubeMP1 package on the MCU, several functions are defined for low-power modes. The table below describes each function defined in STM32CubeMP1 package according to its dedicated low-power mode.

Table 11. Low-power modes functions used on MCU within STM32CubeMP1 package

Low-power mode	Function	Argument	Description
CSleep	HAL_PWR_EnterSLEEPMode	PWR_MAINREGULATOR_ON PWR_LOWPOWERREGULATOR_ON	The regulator parameter is not used for STM32MP15x lines devices and it is kept as parameter just to maintain compatibility with the lower-power families (like STM32L Series)
		PWR_SLEEPENTRY_WFI PWR_SLEEPENTRY_WFE	Specifies if SLEEP mode is entered with WFI or WFE instruction
	HAL_PWR_EnableSleepOnExit	None	Set SLEEPONEXIT bit of SCR register. When this bit is set, the processor reenters SLEEP mode when an interruption handling is over
	HAL_PWR_DisableSleepOnExit	None	Clears SLEEPONEXIT bit of SCR register
CStop, PDDS=0	HAL_PWR_EnterStopMode	PWR_MAINREGULATOR_ON PWR_LOWPOWERREGULATOR_ON	The regulator parameter is not used for STM32MP15x devices and it is kept as parameter just to maintain compatibility with the lower-power families (like STM32L Series)
		PWR_STOPENTRY_WFI PWR_STOPENTRY_WFE	Specifies if Stop mode is entered with WFI or WFE instruction
CStop, PDDS=1	HAL_PWR_EnterStandbyMode	None	This function makes the MCU to go to CStop (WFI) and puts the bit MCU PDDS = 1 allowing the system to go to Standby when the MPU is put in CStandby or in (CStop and MPU PDDS = 1 and MPU CSTBYDIS = 1)

System

The system may enter Stop, LP-Stop, LPLV-Stop or Standby when all the EXTI wakeup sources are cleared and when both the MPU and MCU are in CStop or CStandby mode.

The system enters the Standby mode (power off, clock off) only when the MPU is in CStandby or in CStop with MPU PDDS=1 and MPU CSTBYDIS =1 and when MCU is in CStop with MCU PDDS = 1.

Standby mode entering must be carefully managed by the MCU coprocessor firmware: allowed via MCU “CStop with PDDS=1” only if it is able to manage the Standby state, so at least if it can lose the MCU SRAM content during this period.

It is possible to switch off the MCU clock and peripherals allocated to the MCU (equivalent to a CStop mode) by using the HOLD_BOOT feature. This feature is controlled by the MPU via the BOOT_MCU bit located into the RCC global control register (RCC_MP_GCR). It allows the system to be put in Stop, LPLV-Stop or Standby mode. This feature is activated after a reset of the MCU (when MPU is setting MCURST bit of RCC global reset control set register (RCC_MP_GRSTCSETR)).

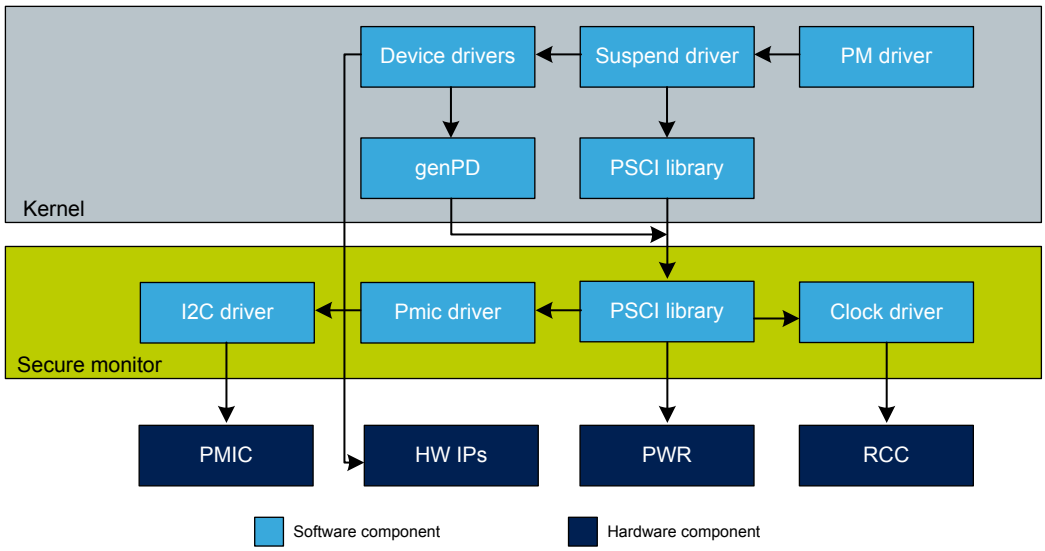
It is also possible to only run code on MPU (MCU in CStop equivalent mode) simply by having MPU not loading any code on MCU side, hence by default when starting the system the MCU is kept inactive (except when boot mode BOOT[2:0] = 0b100).

Note: When entering LPLV-Stop, if VDDQ_DDR is not shutdown, to avoid overconsumption on VDDQ_DDR, the DDR memory must be put in SelfRefresh and DDR PHY must be set in retention mode (setting bit DDRRETEN: DDR retention enable of PWR control register 3 (PWR_CR3) register).

4.5 Power management under Linux

Linux power management is done through the software framework shown in the figure below.

Figure 6. Linux power management software framework



The Linux suspend framework is used to enter the low power modes. It relies on:

- genPD (generic power-domain) framework which allows the definition of power domains and allow the mapping of IPs on those power domains
- PSCI standard to perform the low-level power management process

Additional information can be found on the *Power overview* wiki article.

The secure monitor regulator framework is used to configure the external power-regulator voltages for each power mode and it relies on an ST STPMIC1x (PMIC) driver and an I2C driver.

4.5.1 Linux power commands mapping to STM32MP15x lines power modes

When using Linux operating system, power mode commands are predefined and this section explains how they are mapped to the STM32MP15x lines hardware low-power modes. It also explains the low-power mode control using the MCU coprocessor.

Under Linux, the user can ask the system to enter into low-power mode with the following direct input command:

To enter in Standby mode with DDR off:

```
'shutdown -h 0'
```

or to reach any low-power mode among Stop variants and Standby, with DDR in Self-refresh:

```
echo 'mem' > /sys/power/state
```

The `'disk'`, `'freeze'` and `'standby'` commands are not supported on STM32MP15x lines.

The Cortex®-A7 is put in WFI (wait for interrupt) when entering a low-power mode. **However, the system only reaches the targeted low-power mode after the MCU subsystem has been previously configured in the expected low-power mode.**

For instance the Linux 'mem' command puts the system in Standby mode only if the MCU has been previously set to CStop mode allowing Standby (for example using HAL_PWR_EnterStandbyMode() function).

The list of available wakeup sources is not the same for all low-power modes, so a software mechanism is implemented through the genPD framework to ensure that the low-power mode and the activated wakeup source are consistent.

The strategy is to allow the MPU to enter the deepest low-power mode available according to the currently activated wakeup source(s).

Example 1

The user activates the UART4 as wakeup source. The UART is powered by V_{DDCORE}, so all low-power modes that are modifying V_{DDCORE} are automatically forbidden.

Then, calling `echo mem > /sys/power/state` results in the MPU entering CStop allowing Stop or LP-Stop.

Example 2

The user selects the RTC as wake-up source. The RTC is always powered, whatever the low-power mode status.

Then, calling `echo mem > /sys/power/state` results in the MPU entering CStop allowing Standby.

This mechanism ensures that a blocking situation such as activating UART4 as wakeup source and requesting SoC Standby can never happen.

The system's power mode is the result of both MPU and MCU power states.

The table below lists the deepest possible power mode according to wakeup source groups and lists the equivalence between Linux standard low-power modes and STM32MP15x lines system low-power modes, including the external (Lp)DDR power state (on, off, self refresh (SR)) as it is implemented in the STM32MPU OpenSTLinux Distributions BSP provided environment.

Systems using 32-bit DDR3 interface most likely use external resistance termination on address and command signals. Those systems shall be powered by a resistance termination power supply (VTT) that can be a significant contributor to the overall system consumption during low-power modes.

Hence, by using STPMIC1x, it is possible to switch off this power supply when PWR_ON signals are pulled down (LP-Stop, LPLV-Stop, Standby) and it is preferred to use LP-Stop rather than Stop mode, because LP-Stop mode does not toggle PWR_ON in such use case.

Table 12. Deepest power mode per wakeup source group and equivalence between Linux and STM32MP15x lines

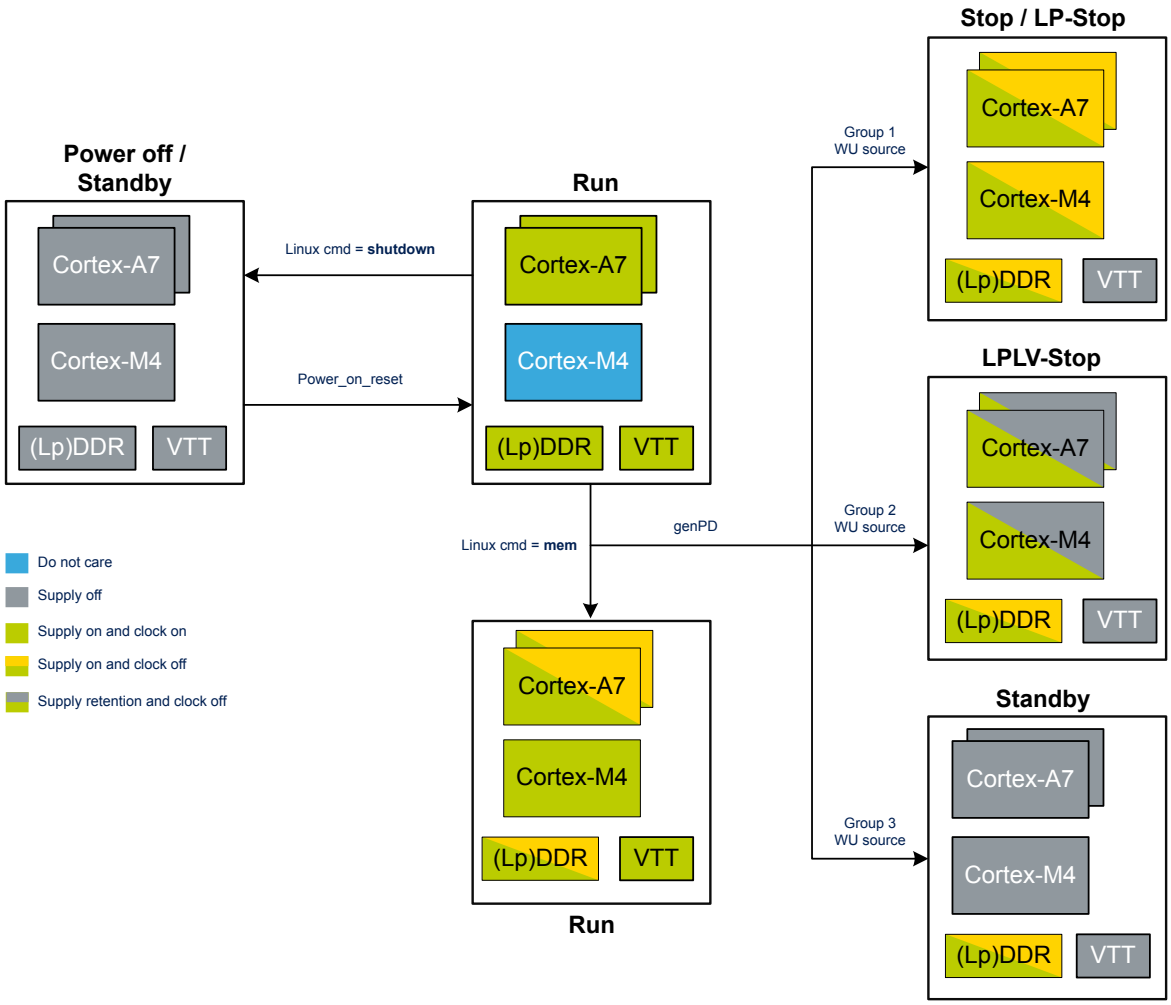
Wakeup source	Linux command	STM32MP15x system deepest power mode	System DDR	Linux kernel state	Power consuming	Wakeup time	Comment / Application guideline
Group 1: USB, CEC, ETH, USART, I²C, SPI	"mem"	Stop or LP-Stop	SR (VTT off)	"Suspend-to-ram"	Medium	Medium	LP-Stop: driving external PWR_LP/PWR_ON permits to design custom strategy for external regulator. Typical application is to switch-off DDR3 termination supply (VTT) in DDR3 32 bit design
Group 2: PVD, AVD, IWDG, GPIO, LPTIM, DTS	"mem"	LPLV-Stop	SR (VTT off)	"Suspend-to-ram"	Low	Medium	LPLV-Stop: save power thanks to power retention. Can be suitable for applications with aggressive power constraints and tolerant with limitations of wakeup source (refer to Table 5. System's low-power mode wakeup capabilities)
Group 3:BOR, Vbat mon, Temp mon, LSE CSS, RTC, TAMP, Wakeup pins	"mem"	Standby	SR	"Suspend-to-ram"	Low	Medium	Standby saves more power at the expense of wakeup time
	"shutdown"	Standby or Off/ VBAT	Off	Shutdown	Very low	High	Wakeup sequence and applicative impact guide the choice between Standby and Off

Note: This table assumes that Cortex®-M4 is in CStop mode for all expected Stop, LP-Stop, LPLV-Stop system power modes and in CStop with PDDS=1 for Standby system power mode.

The following figure shows the power state transitions that are available in the system. The same definitions for each wakeup source group used in the table above are considered in this figure. Group 3 wakeup sources are also functional with Group 2 and Group 1 sources. Group 2 is also compatible with Group 1.

If several wakeup sources are activated in different groups, the low-power mode is chosen by respecting the hierarchy stated in the table above. the choice between Stop or LP-Stop is done through settings in the tree of the secure monitor device tree.

Figure 7. Available power state transitions



4.5.2 Controlling MCU power modes under Linux

There are two possible ways to have the MCU control the power modes under Linux; either using “rpmsg” through inter process communication (IPC) or using “independent clusters”.

“rpmsg” through inter process communication (IPC)

The MPU can be seen as the master and the MCU as a coprocessor. Communication between both processors is done using rpmsg. A Linux kernel is running on the dual core MPU in SMP mode, and a free-rtos is running on the MCU.

Some IPs are shared between the processors and a protection is set in place to ensure the registers consistency. When a core needs to access an IP which is under the responsibility of the other core, it sends a request through rpmsg to get the action done.

Regarding entering/exiting into/from Standby state, before MPU “CStop with PDDS=1 CSTBYDIS=1” state, Linux notifies the MCU that the IPC link is suspended. When waking up from system Standby state with MCU_BEN, the MCU firmware does not communicate with Linux via the IPC. It is only able to send an event (SEV instruction) to the MPU when it needs to wake it up. Then, Linux establishes again the IPC link with the MCU.

Independent clusters

The MCU may manage its low-power mode states in its SW code (no need for rpmsg from MPU).

4.6 Low-power mode exit sequence

Exit from low-power mode (except Standby) can be triggered by interrupt or event depending on how the low-power mode was entered (WFI or WFE). Exit from Standby on MPU can only be triggered by an application reset (like NRST, IWDG), WKUPx pins event or an RTC event.

On exit from low-power mode, it is possible to wake up only MPU, only MCU or both.

- On exit from low-power mode (except Standby) waking up of MPU, MCU or both depends on how the IRQ detection was programmed (visible on MPU, MCU or both).
- On exit from Standby, it is done by setting the two bits MPU_BEN and MCU_BEN in RCC boot control register (RCC_MP_BOOTCR). This configuration is used by the BootROM during the wakeup from Standby. The BootROM then wakesup MPU, MCU or both.

Note: When using STM32MPU OpenSTLinux Distributions, the system always boots only one subsystem: we assume that MCU_BEN and MPU_BEN are exclusive to avoid race conditions that may occur during a parallel start up of both subsystems.

- As a consequence, the MCU should be able to interpret some wake up sources that target the MPU subsystem. For instance, if the wake up is due to an RTC alarm (expected by Linux) then the MCU should wake up the MPU.

When the STM32MP15x devices exit from low-power modes, it is important to make sure all external regulators output voltage are setup to the appropriate level and stable before the Run mode can be applied.

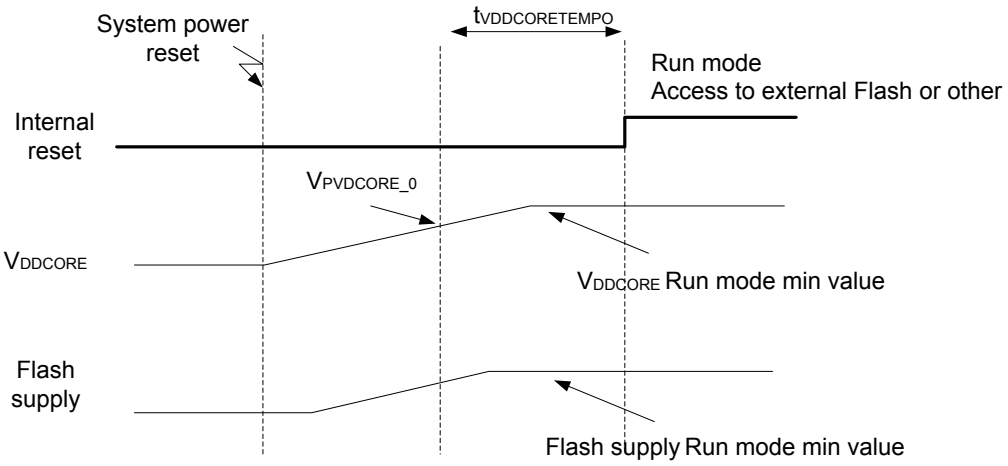
4.6.1 Exit from system power-reset

On system power-reset, the STM32MP15x device monitors the V_{DD} until it reaches its specified VBOR threshold. Also, the STM32MP15x device monitors V_{DDCORE} until it reaches the minimum voltage detector threshold ($V_{PVDCORE\_0}$) after which it is expected that the V_{DDCORE} is correctly established before the $t_{VDDCORETEMPO}$ delay when the internal STM32MP15x device reset signal is deactivated.

In order to find the value of $t_{VDDCORETEMPO}$, refer to the table “Embedded reset and Power Control block characteristics” in the STM32MP15x datasheet ($t_{VDDCORETEMPO}$ minimum value is 200 us). It is expected that the external Flash power supply is ready before STM32MP15x device enters Run mode and reads data from Flash to boot.

- When using STPMIC1x, this is automatically handled since the STPMIC1x does not release the STM32MP15x device NRST pin until all supplies are at their expected value.
- When using a discrete regulator component, the design should make sure those constraints are correctly handled.

Figure 8. **Wakeup sequence from system reset**



4.6.2 Exit from Stop mode

On exit from Stop mode, the voltages are not changed so there is no issue. However, note that while the PLL1 and PLL2 settings are restored thanks to the hardware "clock restore" automatic feature, other PLL have to be reconfigured if needed. MCU is restarted on HSI clock (HSI clock frequency setting is kept similar to the setting prior entering Stop mode). This also applies for all other “Stop” modes (LP-Stop, LPLV-Stop).

4.6.3 Exit from LP-Stop mode

On exit from LP-Stop mode, V_{DDCORE} is not changed, however other voltages at system level might have been switched off or reduced and may need to be set back to their Run mode value.

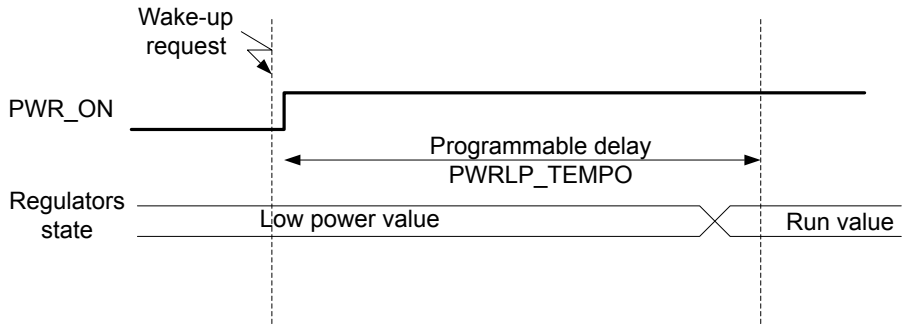
For example the DDR3/DDR3L (x32-bit bus width) termination resistance power supply VTT can be switched off during LP-Stop. The STM32MP15x device waits a programmable delay (PWRLP_TEMPO) in **RCC PWR_LP delay control register** (RCC_PWRLPDLYCR) to allow external regulators Run mode value recovery.

- When using STPMIC1x, the exit from LP-Stop mode is notified from STM32MP15x device by the PWR_ON pin which toggles back to 1 thus requesting changing supply level on STPMIC1x output supplies. The STPMIC1x ensures that the PWRLP_TEMPO can be set to a minimum value of around 1000 us (typical condition).
  - The STM32MP15x maximum output voltage rise time is given by the formula:

    For BUCK output: Max {100 us; V2-V1/3.6 10^{-3} us} (typical)

    with V2: minimum Run mode voltage V1: voltage during low-power mode.
  - For LDO output: ramp-up time is defined by the current limit and total amount of capacitance on LDO output. For 10 µF total output capacitance on LDO3, ramp-up time is 20 µs typical.
  - For a VDD_USB of 3.3 V (from BUCK output), the max rise time would be 3.3/3.6 10^{-3} = 917 us (typical).
  - Exact values in worst case condition must be computed from STPMIC1x datasheet but if wakeup time is not critical it is recommended to use some reasonable margin (for example 5 ms).
- When using discrete external regulator components, the hardware should set the required power supplies to their Run mode value within the PWRLP_TEMPO delay.

Figure 9. **Wakeup sequence from LP-Stop**



4.6.4 Exit from LPLV-Stop mode

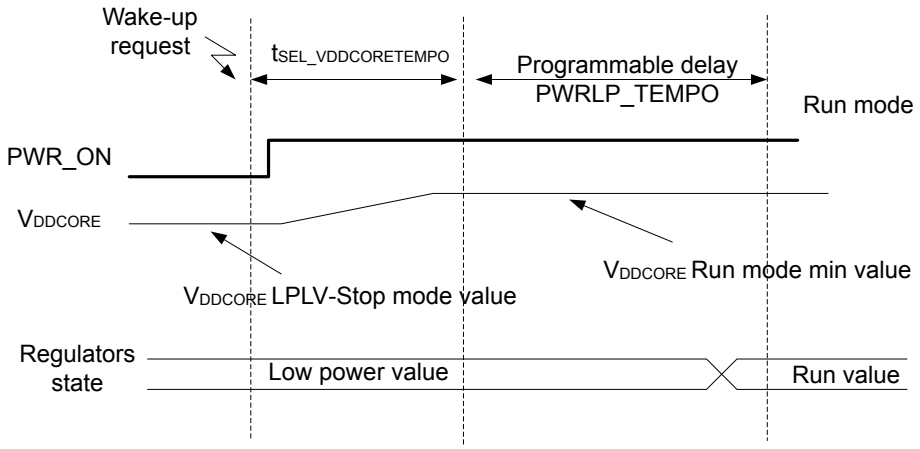
On exit from LPLV-Stop mode, on top of power supplies changed like in LP-Stop mode, the V_{DDCORE} voltage that has been reduced during LPLV-Stop, needs to be set back to its Run mode value.

The STM32MP15x device waits a $t_{SEL\_VDDCORETEMPO}$ delay during which V_{DDCORE} is expected to reach its minimum Run mode value. After this $t_{SEL\_VDDCORETEMPO}$ delay, the STM32MP15x device waits a programmable delay (PWRLP_TEMPO) in **RCC PWR_LP delay control register (RCC_PWRLPDLYCR)** to allow other external regulators Run mode value recovery (as for LP-Stop mode exit sequence described above).

In order to find the value of $t_{SEL\_VDDCORETEMPO}$ refer to the table “Embedded reset and power control block characteristics” of the corresponding STM32MP15x device datasheet ($t_{SEL\_VDDCORETEMPO}$ minimum value is 234 us).

- When using STPMIC1x, the exit from LP-Stop mode is notified from STM32MP15x by the PWR_ON pin which toggles back to 1, thus requesting changing supply level on STPMIC1x output supplies. The STPMIC1x ensures that V_{DDCORE} output supply reaches its minimum Run mode value in less than $t_{SEL\_VDDCORETEMPO}$ delay and that PWRLP_TEMPO can be set to a minimum value defined by the same formula provided in Section 4.6.3 Exit from LP-Stop mode.
  - If during LPLV-Stop V_{DDCORE} power supply (from BUCK output) is reduced to 0.9 V then V1= 0.9 V, V2= 1.2 V so maximum rise time is 0.3 / 3.6 10^{-3} = 83 us (typical). The worst condition value should be computed from the STPMIC1x datasheet and should be less than $t_{SEL\_VDDCORETEMPO}$ (234 us).
  - Same formula to be used for other power supplies that should have rise time (in worst condition) less than PWRLP_TEMPO.
  - If wakeup time is not critical it is recommended to use reasonable margin (for example 5 ms).
- When using discrete external regulator components, the hardware should be able to set the V_{DDCORE} supply to its minimum Run mode value within the $t_{SEL\_VDDCORETEMPO}$ delay and should set the other required power supplies to their Run mode value within the PWRLP_TEMPO delay.

Figure 10. **Wakeup sequence from LPLV-Stop**



4.6.5 Exit from Standby mode

On exit from Standby mode, on top of power supplies changed like in LP-Stop mode, V_{DDCORE} voltage (that has been turned Off during Standby mode) needs to be switched On and set to its Run mode value.

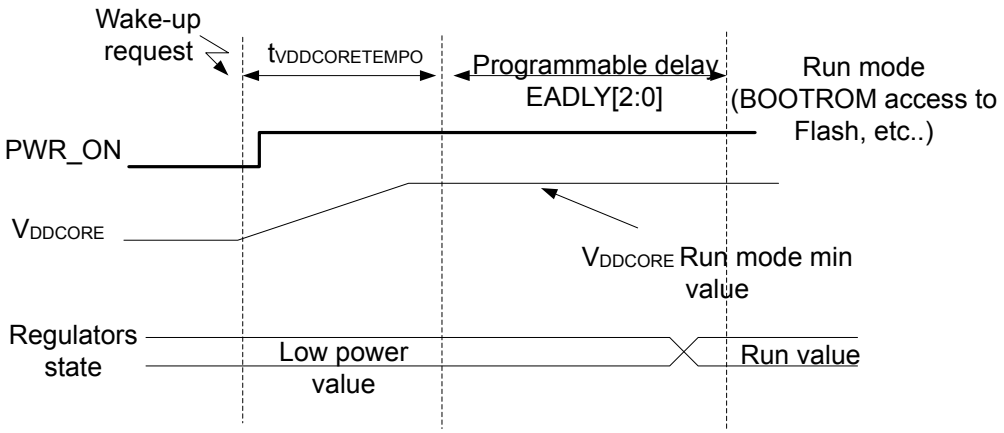
During Standby the V_{DD} supply is kept active and so at wakeup from Standby, the pin PWR_ON is set back to 1 (part of the digital logic in the PWR controller is supplied by VDD and so can drive PWR_ON pin).

The STM32MP15x device waits a $t_{VDDCORETEMPO}$ delay during which V_{DDCORE} is expected to reach its minimum Run mode value. The STM32MP15x device then waits a programmable delay (EADLY[2:0]) in **RCC reset duration and LSI control register (RCC_RDLSICR)**, before the BOOTROM performs any access to the external memories to allow all required external power regulators Run mode value recovery (external Flash power supplies or other).

In order to find the value of $t_{VDDCORETEMPO}$ refer to the table “Embedded reset and power control block characteristics” in the corresponding STM32MP15x device datasheet ($t_{VDDCORETEMPO}$ minimum value is 200 us).

- When using STPMIC1x, the exit from Standby mode is notified from STM32MP15x device by the PWR_ON pin which toggles back to 1 thus requesting changing supply level on STPMIC1x output supplies. The STPMIC1x device ensures that V_{DDCORE} output supply reaches its minimum Run mode value in less than $t_{VDDCORETEMPO}$ delay and that the EADLY[2:0] delay can be set to a minimum value defined by the same formula provided in Section 4.6.3 Exit from LP-Stop mode. It is recommended to use a reasonable margin of 5-10 ms for EADLY[2:0].
- When using discrete external regulator components, the hardware should be able to set the V_{DDCORE} supply to its minimum Run mode value within the $t_{VDDCORETEMPO}$ delay and should handle the setting of the required power supplies to their Run mode value within the programmed EADLY[2:0] delay.

Figure 11. Wakeup sequence from Standby



Many system configuration are reset after a wake up from Standby:

- The clock tree is back to HSI source, so the retention firmware (if present) should always use the HSI as kernel clock for the peripherals that it needs to use.
- It is recommended that the retention firmware stays on HSI in order to avoid clock tree conflict with Linux (if used).

4.6.6 Exit from VBAT mode

When the STM32MP15x device is in VBAT mode, only the VSW supply is maintained by an external VBAT supply.

The available wakeup sources (TAMP, RTC) can activate PC13 pin to inform external regulator about the wakeup request. The external regulator then restarts the power sequencing like for a system power reset.

4.6.7 Exit from MPU CStop and CStandby

The difference between MPU CStop (PDDS=1, CSTBYDIS=1) and CStandby is that the exit mode is different (see Table 6. Power-system mode versus sub-system mode summary).

When exit from MPU CStandby with System in Run, Stop, LP-Stop or LPLV-Stop, program execution restarts with a local MPU reset. This "reboot" can affect MCU operation due to possible reinitialization of some peripheral registers by the BootROM when reloading MPU initialization code from external Flash. Refer to Table 6. Power-system mode versus sub-system mode summary.

This "reboot" cannot be correctly executed in case the external Flash memory is in a mode not expected by the BootROM (for example if the SD-Card has been set in UHS-I mode while BootROM is expected to be in standard mode). For those reasons, it is usually preferably to use the MPU CStop mode rather than the MPU CStandby mode. Note that MPU Cstandby mode is not used in the ST Linux distribution.

When exit from MPU CStandby or CStop (PDDS=1, CSTBYDIS=1) with System in Standby, program execution restarts in the same way as after a power on reset (such as option bytes loading or reset vector fetched). Refer to Table 6. Power-system mode versus sub-system mode summary. In such exit mode it is expected that the MCU restarts from reset and that there is no impact from MPU reboot.

Note: MPU CStandby is not used on ST Linux distribution since its wake up is not deterministic (CStandby exit is done without power cycle whereas Standby exit implies a power cycle).

Refer to the application note *Getting started with STM32MP1 Series hardware development* (AN5031) for more details on hardware constraints.

When exit from MPU CStop with system in Stop, LP-Stop or LPLV-Stop mode, program execution restarts through the interrupt handler, (if the WFI (wait for interrupt) instruction or Return from ISR was used to enter into low-power mode) or restarts from the instruction following WFE (wait for event) (if the WFE instruction is used to enter into low-power mode). This is the recommended mode to ensure independent MCU behavior if MPU is restarted before system Standby entry.

5 STM32MP15x lines peripherals configuration for low-power

STM32MP15x lines implement many peripherals. Their configuration can play an important role in the power consumption. This section describes how to configure the relevant peripherals.

5.1 Peripheral assignment and allocation

There are three available context during Run time:

- Cortex-A7 secure
- Cortex-A7 non-secure
- Cortex-M4

Each peripheral can be “assigned” to one or several (shared) of those contexts. This is done by the Extended TrustZone protection controller (ETZPC).

Each peripheral is assigned at reset time (A7, A7 secure, M4 or shared) according to the Reference manual register content, or during boot time when using STM32MPU OpenSTLinux Distributions (see Figure 13. STM32MP15x lines peripheral assignment using opentSTLinux distribution). The software can later update this default configuration during runtime.

When one of the context wants to use a peripheral assigned to it, it has to ensure the peripheral is clocked. So prior to use a peripheral, the MPU or MCU has to enable it, also it can define if this peripheral remains active in CSleep mode. This clocking enabling is done by the RCC and is called “allocation” of peripheral to a processor.

- The MPU can allocate or deallocate a peripheral for itself and for the MCU.
- The MCU can allocate or deallocate a peripheral for itself but not for the MPU.

Enabling a peripheral is done by setting the dedicated PERxEN bit on register RCC_MP_xxxxENSETR on MPU side and register RCC_MC_xxxxENSETR on MCU side.

Disabling a peripheral is done by setting the dedicated PERxEN bit on register RCC_MP_xxxxENCLRR on MPU side and register RCC_MC_xxxxENCLRR on MCU side.

Activating a peripheral during CSleep is done by setting the dedicated PERxLPEN bit on register RCC_MP_xxxxLPENSETR on MPU side and register RCC_MC_xxxxLPENSETR on MCU side.

Disabling a peripheral during CSleep is done by setting the dedicated PERxLPEN bit on register RCC_MP_xxxxLPENCLRR on MPU side and register RCC_MC_xxxxLPENCLRR on MCU side.

When using STM32Cube_FW_MP1 the user can allocate the peripheral and control the peripheral clocks in CSleep mode through the dedicated function defined in STM32MP15xx_hal_rcc.h:

```
__HAL_RCC_xxxx_CLK_ENABLE();
__HAL_RCC_xxxx_CLK_SLEEP_ENABLE();
```

On the MPU side, the IP drivers enable their clocks by making the allocation. Refer to the sections “Peripheral allocation” and “General clock concept overview” of the corresponding reference manual (see Table 2. STM32MP15x lines configuration) for more details.

For most of the peripherals, once 'allocated' to a processor, it is possible for the application to control the activation/deactivation of their kernel and bus interface clocks.

The peripheral allocation is used by the RCC to automatically control the clock gating according to the CPUs modes.

Depending on the operating power supply range, some peripheral may be used with limited functionality and performance. For more details refer to the section “General operating conditions” in the corresponding datasheet.

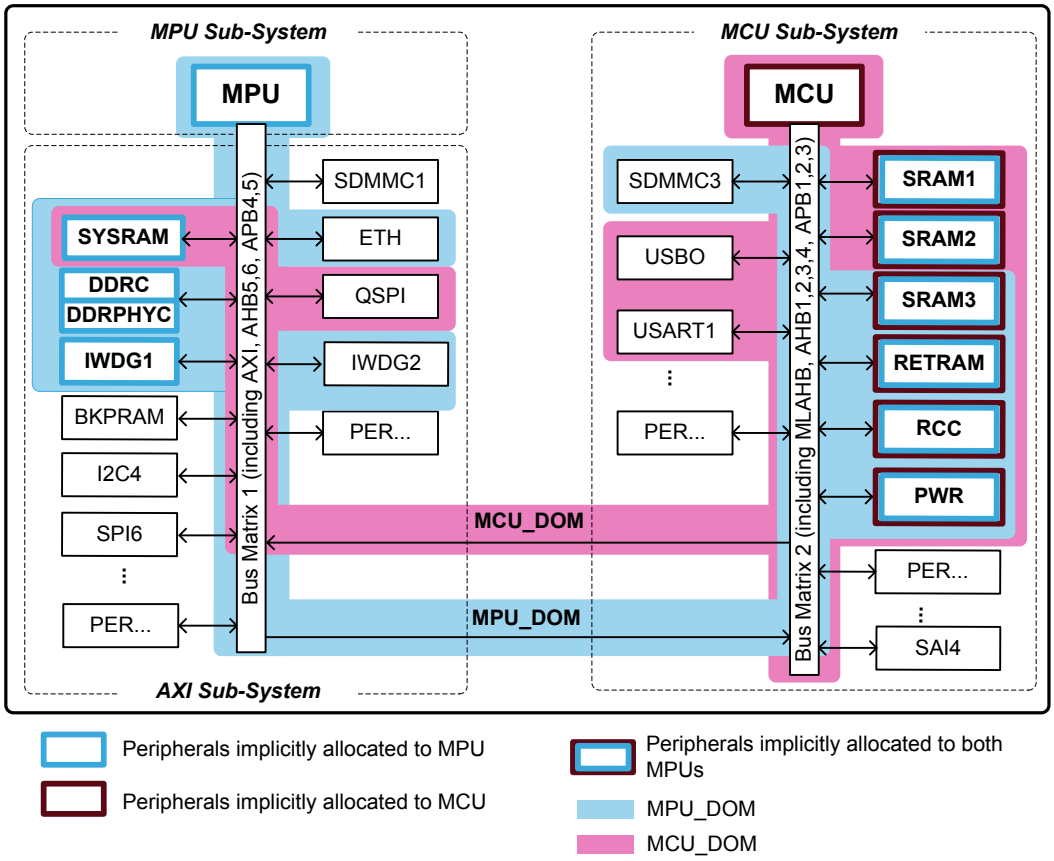
The figure below gives an example of peripheral allocation with the following considerations:

- The MPU enabled the ETH, SDMMC3, SRAM3 and RETRAM. Note that SYSRAM, DDRC, DDRPHYC and IWDG1 are implicitly allocated to MPU. The group composed by the MPU, bus matrix 1, bus matrix 2, and peripherals allocated via RCC_MP_xxxx registers are forming the MPU peripheral domain (MPU_DOM).
- The MCU enabled SYSRAM, QSPI, USBO and USART1. The group composed by the MCU, bus matrix 1, bus matrix 2 and peripherals allocated via RCC_MC_xxxx registers are forming the MCU peripheral domain (MCU_DOM).

Note: PWR and RCC are common resources and implicitly allocated to both MPU and MCU. SRAM1, SRAM2, SRAM3 and SRAM4 are also implicitly allocated to both MPU and MCU because the MLAHB bridge is enabled if one of the processor is in CRun. RETRAM is also implicitly allocated to both MPU and MCU in order to allow accesses when the system exits from Standby.

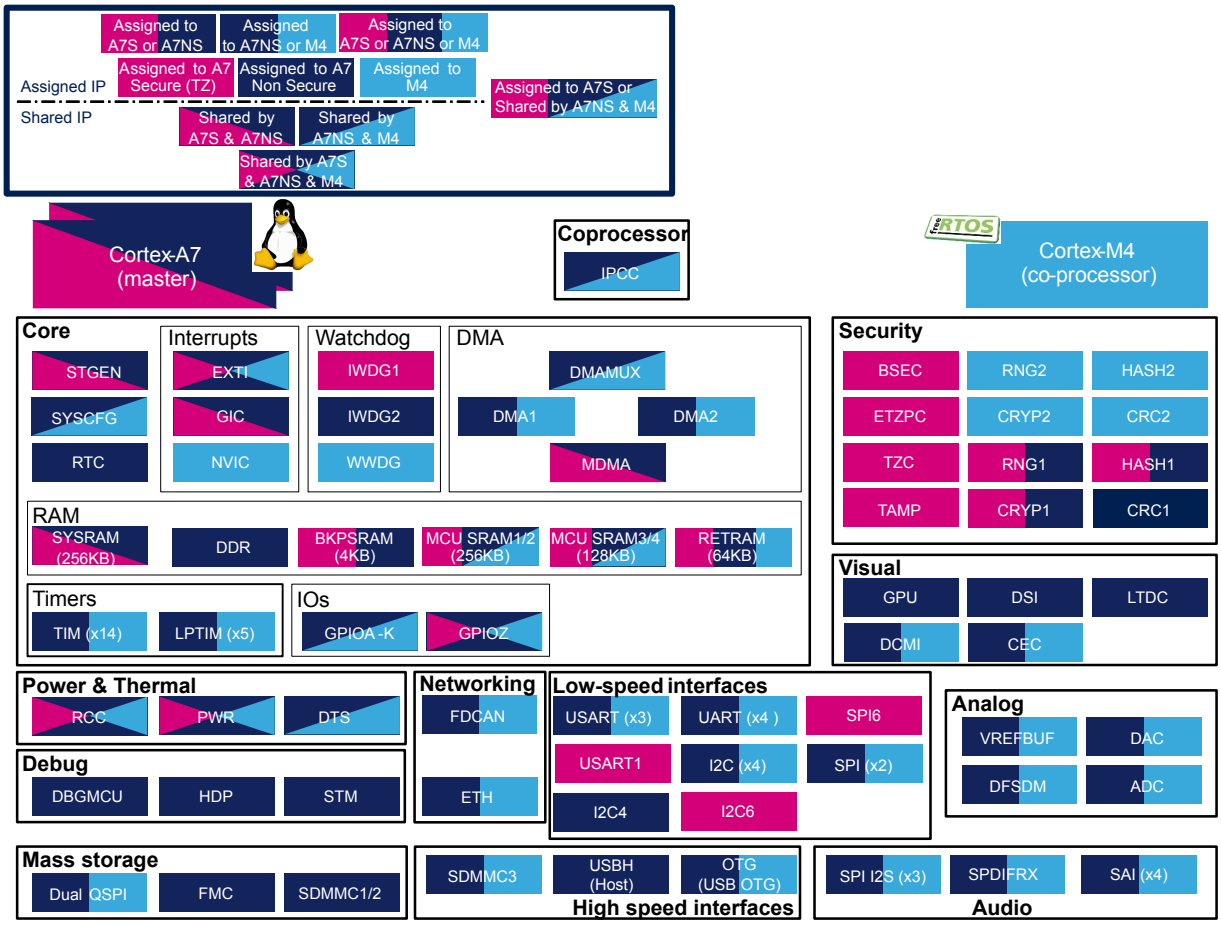
Note: The BKPSRAM has a dedicated enable bit in order to gate the bus interface clock. Prior to use the enable bit, the BKPSRAM needs to be enabled.

Figure 12. Peripheral allocation example



If using ST Linux distribution, the figure below shows the list of peripherals and their possible assignment to Cortex-A7 secure, Cortex-A7 non-secure, Cortex-M4 or Shared context.

Figure 13. STM32MP15x lines peripheral assignment using opentSTLinux distribution



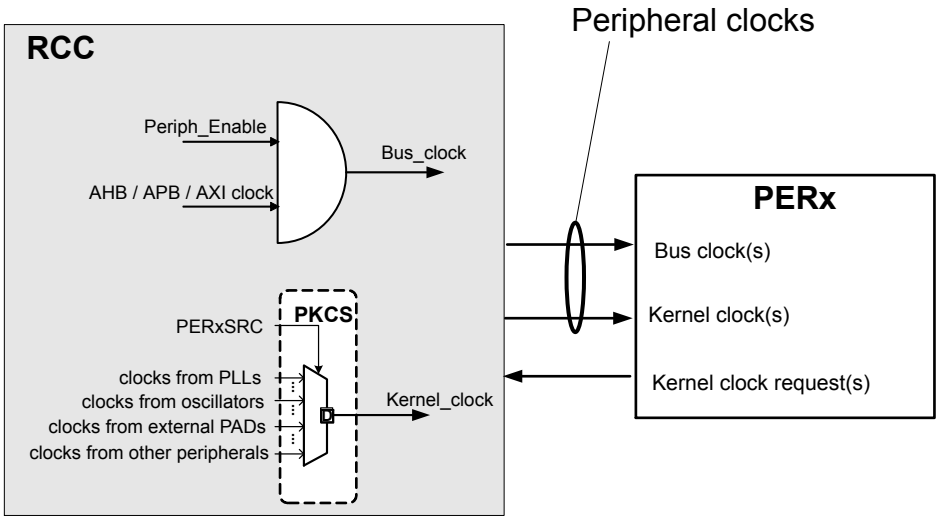
5.2 Peripheral clock distribution

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

- The bus interface clocks.
- The kernel clocks.

The following figure describes the peripheral clock distribution on the STM32MP15x devices.

Figure 14. 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).

When using STM32CubeMP1 package kernel clock of the peripheral can be chosen by using the dedicated function defined in stm32mp1_xx_hal_rcc_ex.h file:

```
__HAL_RCC_PERx_CONFIG (__PERxCLKSource__);
```

On MPU side, the Linux clock driver offers a 'clk_set_parent' service able to select the kernel clock.

For more details about kernel clock distribution and peripheral clock gating refer to the section “Peripheral clock gating control” of the corresponding reference manual (see Table 2. STM32MP15x lines configuration).

5.3 Stop, LP-Stop, LPLV-Stop peripheral allocation

In Stop, LP-Stop mode, the peripherals using the LSI or the LSE clock and peripherals having a kernel clock request selecting a clock source in the RCC which is able to run in Stop mode, are still able to operate.

In order to allow the I2Cs or U(S)ARTs to work in CStop or system Stop, LP-Stop mode, the user must select an oscillator as kernel clock: hse_ker_ck, hsi_ker_ck or csi_ker_ck according to RCC PERxEN. The selected oscillator(s) is provided to the peripheral when the peripheral generates a kernel clock request.

The oscillator(s) remain activated in Stop, LP-Stop mode if the xxxKERON=1, allowing immediate delivery of the clock on wakeup from Stop, LP-Stop or on peripheral kernel clock request.

Refer to Table: Functionalities depending on system operating mode in the corresponding reference manual (see Table 2. STM32MP15x lines configuration).

In LPLV-Stop mode, the external power supply is expected to lower its voltage level. As a result, only a limited number of peripheral are still functional (BOR, PVD, AVD, VBATH-L monitoring, TEMPH-L monitoring, LSI, LSE, LSE CSS, RTC, IWDG), additionally some can wake up the system (DTS, GPIO). Refer to Table: Functionalities depending on system operating mode in the corresponding reference manual (see Table 2. STM32MP15x lines configuration).

6 Debug tips

This section describes how to use the debug features in low-power modes.

6.1 Enabling debugging in low-power mode: low-power mode emulation

When entering a low-power mode, the clock of the processors, the subsystem clocks or the core power supply may be switched off. This action prevents debug accesses to the related un-clocked domains. The STM32MP15x lines devices support a low-power mode emulation that keeps debug capability while in low-power mode. This is controlled by DBGSLEEP, DBGSTOP and DBGSTBY bits in DBGMCU configuration register (DBGMCU_CR).

Note: DBGMCU set of registers applies to both MPU and MCU subsystem. The naming DBGMCU has been kept for consistency with legacy STM32 MCU products.

Comprehensive information can be found in the following parts of the corresponding STM32MP15x device reference manual (see Table 2. STM32MP15x lines configuration):

- Low-power emulation mode in RCC chapter.
- Microcontroller debug unit (DBGMCU) in the Debug support chapter.

Note: Using low-power emulation modes has a significant impact on the power savings (various clocks and power supply being kept on). Such modes should only be used for debugging purposes. Power consumption measurements are not relevant while using low-power mode emulation.

Table 13. HAL debug function

Function	Description
HAL_DBGMCU_EnableDBGSleepMode() HAL_DBGMCU_DisableDBGSleepMode()	Enable/disable the Debug module during CSleep mode
HAL_DBGMCU_EnableDBGStopMode() HAL_DBGMCU_DisableDBGStopMode()	Enable/disable the Debug Module during Stop modes
HAL_DBGMCU_EnableDBGStandbyMode() HAL_DBGMCU_DisableDBGStandbyMode()	Enable/disable the Debug Module during Standby mode

6.2 Using HDP for debugging the low-power modes

The hardware debug port (HDP) allows the observation of internal signals that can be used to debug the entry in and the exit from low-power modes. Those signals can be output on some pins of the STM32MP15x device and easily monitored (though a scope or a logic analyzer). See the *Hardware debug port* section in the corresponding STM32MP1 Series reference manual (see Table 2. STM32MP15x lines configuration) for the comprehensive list of signal and their mixing configuration.

HDP and associated GPIOs should be disabled in the final application in order to reach a lower power consumption.

The most important signals for power-mode debugging are described in the tables below.

Table 14. Monitoring the CSleep modes

Signal	Description
CA7 STANDBYWFI0 (HDP7, HDP_MUX=0) CA7 STANDBYWFI1 (HDP6, HDP_MUX=0) CA7 STANDBYWFE0 (HDP7, HDP_MUX=1) CA7 STANDBYWFE1 (HDP6, HDP_MUX=1)	The CA7 STANDBYWFI0 and CA7 STANDBYWFI1 (in case of WFI) or CA7 STANDBYWFE0 and CA7 STANDBYWFE1 (in case of WFE) signals can be used to monitor the CA7 CSleep mode. Those signals indicate whether a core is in WFI or WFE state.
CM4 SLEEPING (HDP4, HDP_MUX=1)	The CM4 SLEEPING signal can be used to monitor the CM4 CSleep mode (WFI or WFE).

Table 15. Monitoring the entry in CStop modes and in Stop mode

Signal	Description
RCC pwrds_mcu (HDP2, HDP_MUX=6) RCC pwrds_mpu (HDP1, HDP_MUX=6)	The RCC pwrds_mcu (or RCC pwrds_mpu) signal can be used to monitor the CM4 CStop mode (or CA7 CStop mode).
RCC pwrds_sys (HDP3, HDP_MUX=6)	The RCC pwrds_sys signal can be used to monitor the STM32MP15x device Stop mode.

Table 16. Monitoring the exit from Stop mode

Signal	Description
pwrwake_sys (HDP0, HDP_MUX=0)	The pwrwake_sys signal can be used to monitor that the EXTI has received a wakeup interrupt from a peripheral.
pwrwake_mpu (HDP2, HDP_MUX=0) pwrwake_mcu (HDP1, HDP_MUX=0)	The pwrwake_mpu (or pwrwake_mcu) signal can be used to monitor that the CA7 subsystem (or CM4 subsystem) is requested to return to CRun mode.

Refer to http://wiki.st.com/stm32mpu/wiki/HDP_Linux_driver for an example of HDP usage.

Bare metal HDP initialization sample code

```
void HPDConfig(void) {
GPIO_InitTypeDef GPIO_InitStruct;
/*
* HDP AF2 (AF0)
* 0 PI12 (PA4)
* 1 PI13 (PC6)
* 2 PJ5 (PE13)
* 3 PJ6 (PE15)
* 4 PK1 (PC7)
* 5 PK2 (PD3)
* 6 PK5 (PB8)
* 7 PK6 (PB9)
*/

/* Clock GPIOI GPIOJ and GPIOK */
__HAL_RCC_GPIOI_CLK_ENABLE();
__HAL_RCC_GPIOJ_CLK_ENABLE();
__HAL_RCC_GPIOK_CLK_ENABLE();
/*** Route HDP outputs to GPIOs ***/
/* AF2,PI12 -> HDP0 */
/* AF2,PI13 -> HDP1 */
GPIO_InitStruct.Pin = GPIO_PIN_12|GPIO_PIN_13;
GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Alternate = GPIO_AF2_HDP;
HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);

/* AF2,PJ5 -> HDP2 */
/* AF2,PJ6 -> HDP3 */
GPIO_InitStruct.Pin = GPIO_PIN_5|GPIO_PIN_6;
HAL_GPIO_Init(GPIOJ, &GPIO_InitStruct);

/* AF2,PK1 -> HDP4 */
/* AF2,PK2 -> HDP5 */
/* AF2,PK5 -> HDP6 */
/* AF2,PK6 -> HDP7 */
GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_2|GPIO_PIN_5|GPIO_PIN_6;
HAL_GPIO_Init(GPIOK, &GPIO_InitStruct);

/* Clock HDP */
__HAL_RCC_HDP_CLK_ENABLE();

/*** Select signals for HDPx outputs ***/
/* Select CM4 SLEEPDEEP signal as the HDP0 output */
MODIFY_REG(HDP->HDP_MUX, HDP_MUX_MUX0, (HDP_MUX_MUX0_1 ));
/* Select RCC pwrds_mpu signal as the HDP1 output */
MODIFY_REG(HDP->HDP_MUX, HDP_MUX_MUX1, (HDP_MUX_MUX1_6 ));
/* Select RCC pwrds_mcu signal as the HDP2 output */
MODIFY_REG(HDP->HDP_MUX, HDP_MUX_MUX2, (HDP_MUX_MUX2_6 ));
/* Select RCC pwrds_sys signal as the HDP3 output */
MODIFY_REG(HDP->HDP_MUX, HDP_MUX_MUX3, (HDP_MUX_MUX3_6 ));
/* Select CM4 SLEEPING signal as the HDP4 output */
MODIFY_REG(HDP->HDP_MUX, HDP_MUX_MUX4, (HDP_MUX_MUX4_1 ));
/* Enable HDP */
SET_BIT(HDP->HDP_CTRL, HDP_CTRL_EN);
}
```

6.3 Linux debug hints

Checking clock summary using: cat /sys/debug/kernel/clk/clk_summary.

- It displays the clock tree in a hierarchical way with frequency and state.

Monitoring the power status flags using ‘devregs’ command.

- PWR_MPUCR: contains the Stop flags for MPU.
- PWR_MCUCR: contains the Stop flags for MCU.
- RCC_MP_RSTSR: contains the reset reasons including Standby and Cstandby.

If the system does not wakes up, check that at least one wakeup source is enabled, that wakeup peripherals are clocked from HSI or HSE clock and that wakeup pin polarity and pull up/pull down have been set.

If the system does not enter low-power:

- Check that a wakeup event is not already pending when calling the mode.
- cat /proc/interrupts can give an indication of such events.
- Check EXTI setup.

If overall power consumption is too high:

- Check that clocks are released when a peripheral is not used. This can be checked thanks to clk_summary command.

Revision history

Table 17. **Document revision history**

Date	Version	Changes
30-Jan-2019	1	Initial release.
11-Feb-2019	2	Updated: • Section *Linux power commands mapping to STM32MP15x lines power modes* • Table *Deepest power mode per wakeup source group and equivalence between Linux and STM32MP15x lines*
18-Feb-2019	3	Updated Table *System low-power modes summary*
03-Nov-2020	4	Updated: • Replaced "STM32MP1 Series" for "STM32MP15x lines" in the whole document to refer to the applicable products of this application note. • Document's title and cover page. • Section 3.2 System supplies (VDD and VDDCORE) • Figure 2. Discrete supplies example 3.3V I/Os with DDR3L • Figure 3. STPMIC1x example 3.3 V I/Os with DDR3L • Table 5. System's low-power mode wakeup capabilities • Section 4.3 External control signals PWR_ON, PWR_LP pins • Section 4.5 Power management under Linux • Section 4.5.1 Linux power commands mapping to STM32MP15x lines power modes • Section 6.2 Using HDP for debugging the low-power modes
20-Sep-2022	5	Updated: • Table 5. System's low-power mode wakeup capabilities • Table 12. Deepest power mode per wakeup source group and equivalence between Linux and STM32MP15x lines Replaced TEMP with DTS in: • Section 5.3 Stop, LP-Stop, LPLV-Stop peripheral allocation • Table 5. System's low-power mode wakeup capabilities • Table 3. Glossary Added TEMP in Table 12.

Contents

List of tables

List of figures