TN1271
Technical note

SPC58xB/C/G/H Low Power Modes

Introduction

The technical note describes the Low Power Modes available on the SPC58xB/C/G/H devices family. It is focused and applied to all derivatives of the 1M, 2M, 4M, 6M and 10M dies. The intention of this technical note is mainly to clarify the proper usage case and allowed configuration sequences preventing potential issues during mode entering and mode transitions.

The technical note assumes that user is already familiar with basic principles and experienced with Mode Entry module configuration.

SPC58xB/C/G/H devices provide the Mode Entry module (MC_ME) which controls the microcontroller mode and mode transition sequences in all functional states. It contains configuration, control and status registers accessible for the application. Please refer to the particular SPC58xB/C/G/H derivative reference manual for a register interface detailed description.

Contents

List of tables

List of figures

1 Device summary

Device names families referenced within this technical note are mapped according to the *Table 1*.

Table 1. Device names mappings

Flash size	Derivative markings
1M	SPC582Bx
2M	SPC584Bx
4M	SPC584Cx, SPC58ECx
6M	SPC584Gx, SPC58EGx, SPC58NGx
10M	SPC58NHx. SPC58EHx

Note: By default SPC58xGx means cut 2.0 and higher (marking BA on the package) while previous cut 1.1 means SPC58xGx (marking AB on the package). They are slightly different. (STANDBY RAM start address, only INTC_1, sram0_size = 128 K...). For details please refer to particular derivative RM and internally attached IO Definition xls file.

All the current consumptions reported in this technical note are based on a 5 V power supply and an ambient temperature of 25 ° C, the consumptions are measured on the ST EVB mini module outside the motherboard while the debugger is disconnected.

2 Device modes overview

The Mode Entry module (MC_ME) is a SPC58xB/C/G/H module that allows the user to centralize the control of all device modes and related modules / parameters within a unique module.

SPC58 modes can be subdivided in the following groups:

- SYSTEM modes:
  All the modes (RESET, TEST, SAFE, DRUN) in which the device must be initialized and properly configured.
- RUNNING modes:
  All the modes (RUN0:3) used to obtain the full device performance. Clock distribution configuration and clock dividers could be used to optimize the overall power consumption.
  The device supports WAIT instruction to stop the core with the capability to restart with very short latency.
- Low power modes:
  All the modes (HALT, STOP, STANDBY) used to minimize the power consumption. STANDBY allows to reach minimal consumption. Smart standby wake-up Unit (SSWU) additionally allows to use a sequence of user programmable tasks without CPU intervention.

3 Device modes and consumption reduction

Generally SPC58 architecture allows to decrease power consumption via configurable MCU clock tree and configurable power domains.

Configurable clock tree allows to enable or disable clocks for particular peripherals or apply clock dividers which decreases dynamic consumption depending on clock frequency. Please refer to RM Clock generation diagram to see all clock configuration possibilities.

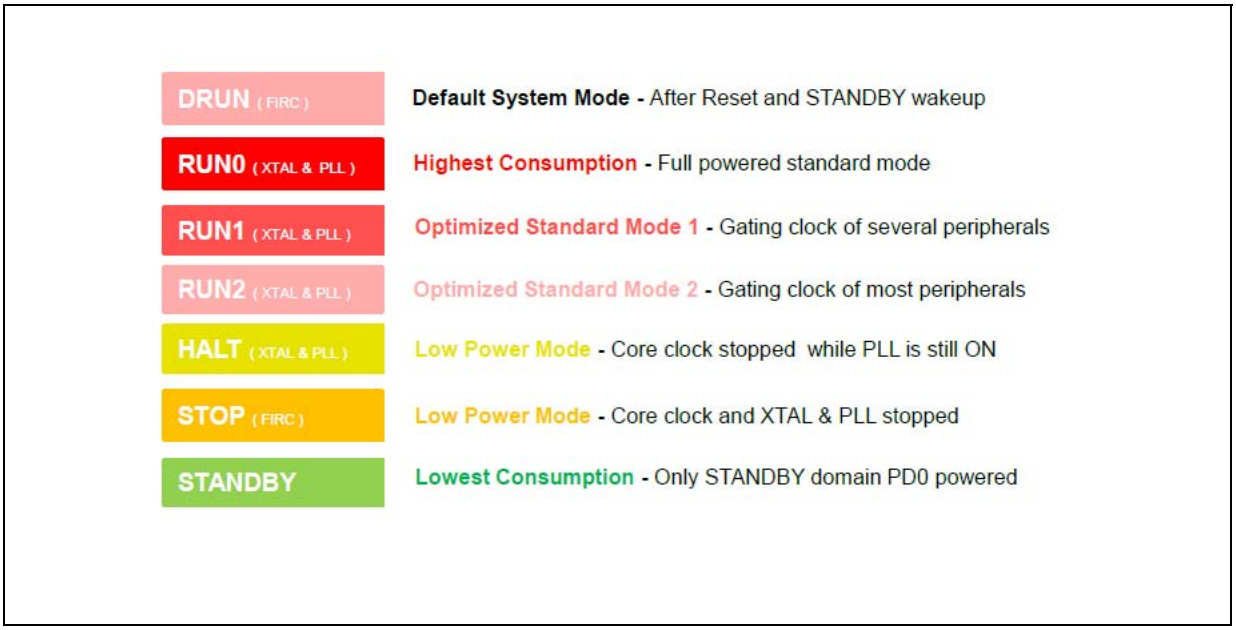
Power segmentation allows (via PD0/1/2/3 power domains) to disconnect particular power domains from power and significantly decrease total current leakage.

Using Device Modes user can configure and switch between particular consumption profiles. We can consider the following levels from the smallest consumption reduction to the highest:

- RUN0/1/2/3 modes allow to selectively configure particular peripherals to be clocked / gated and select clock source (FIRC, XTAL, PLL).
- Higher consumption reduction can be reached using HALT mode where all clock sources can be still kept running but core clock is stopped. Wake-up is very fast and performed via any interrupt or wake-up hardware event. All configurations are kept and execution continues from stopped place.
- To reach even higher consumption reduction we can use STOP mode which does not allow to keep PLL clocks running. Wake-up is slower and performed via any interrupt or wake-up hardware event. All configurations are kept and execution continues from stopped place.
- For the highest power reduction we can use STANDBY mode which is based on power domain disconnection and requires reconfiguration after wake-up. Wake-up can be performed only via wake-up events and is similar to the reset.

Example of particular configuration use case:

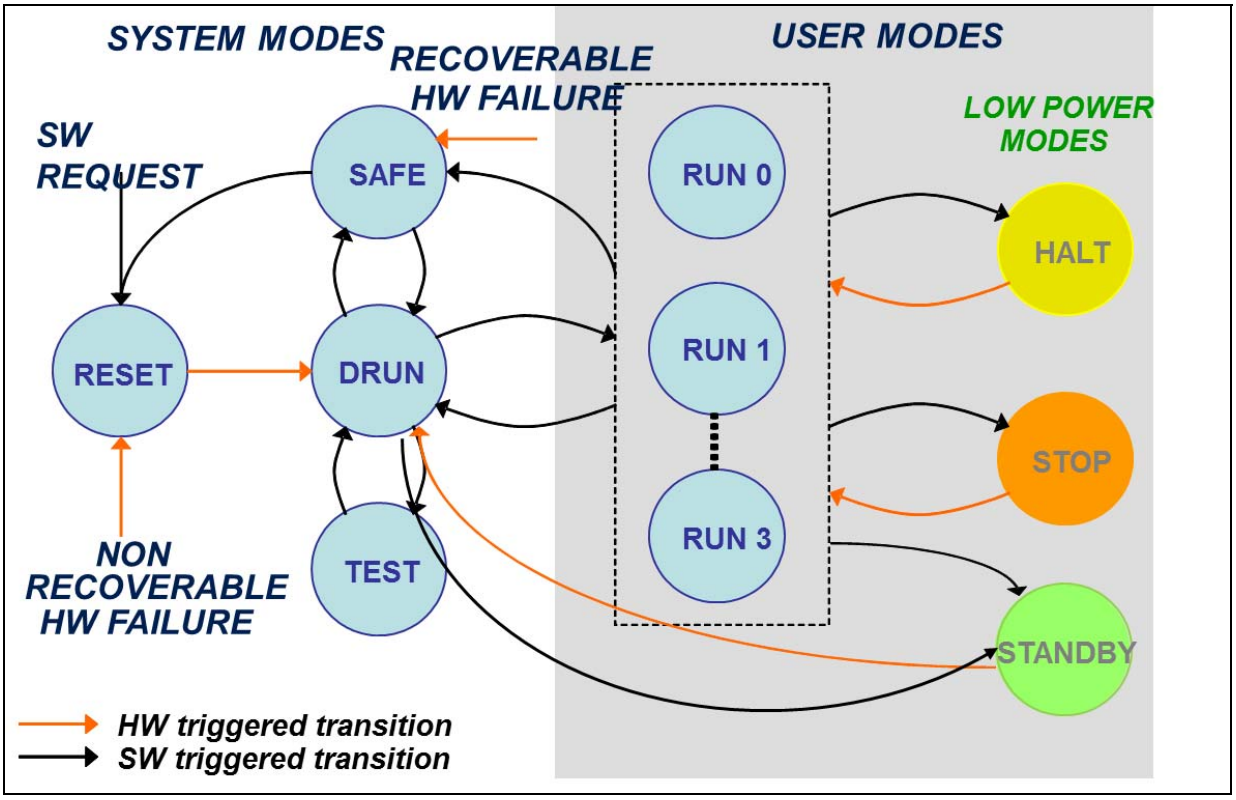
Figure 1. Example of particular device mode configuration



4 Mode transition diagram

The Mode Entry module (MC_ME) controls mode transitions. Entering low power mode is performed as SW triggered transition while exiting low power mode is automatically managed by MCU hardware triggered by wake-up events. The below diagram shows all possible mode transitions.

Figure 2. Mode Transitions diagram



4.1 Entering low power mode

It could be performed as SW triggered transition.

It is triggered by SW writing ME_MCTL register twice:

1st write: TARGET_MODE + KEY (0x5AF0)

2nd write: TARGET_MODE + INVERTED KEY (0xA50F)

Transition completion signalling: status bit / interrupt

while (ME.GS.B.S_MTRANS);

It is recommended to implement the instructions above while looping with software timeout.

Reasons while SW triggered transition was not successfully performed:

- HALT and STOP
  - While any interrupt or wake-up event is active, then MCU mode does not change (no error)
  - In case of invalid configuration or not allowed mode transition then consequently status error flag or invalid mode interrupt is triggered
- STANDBY
  - While any wake-up event is active, then MCU mode does not change (no error)
  - In case of invalid configuration or not allowed mode transition then consequently status error flag or invalid mode interrupt is triggered

4.2 Exiting low power mode

It is HW triggered transition.

It is automatically managed by MCU hardware via wake-up events.

The list of possible hardware low power mode wake-up events (sources) is described in *Table 2*.

Table 2. List of wake-up sources

	Wake-up	Interrupt	Note	Return
HALT	√	√	—	Came back to the previous mode from which entered in (RUN0,1,2,3)
STOP	√	√	IF SYSTEM CLOCK = ON	Came back to the previous mode from which entered in (RUN0,1,2,3)
STANDBY	√	—	API / SSWU, RTC, WKUP LINES	DRUN

4.3 Low power modes wake-up times

User has to consider different wake-up times of the low power modes. Faster wake-up time exhibits HALT mode while the longest wake-up time exhibits STANDBY mode. HALT, STOP low power modes control current consumption mainly by gating clocks to particular modules while original configuration and status are kept during LP mode.

STANDBY mode is using a different approach switching most of MCU modules from power supply and consequently these modules have to be initialized again after wake-up.

On the other side entering any low power mode we can consider it as very fast.

Table 3. STANDBY wake-up times

Wake-up into ram	Wake-up into FLASH
125 µs	200 - 500 µs(1)

1. According to device DCF configuration (500 µs if HSM handshaking is activated).

4.4 Error low power modes management

The following rules have to be met to avoid invalid mode configuration or transition:

- Preventing Invalid Mode Configuration
  - IRCON should be ON if: SYSCLK = RC_CLK
  - XOSCON should be ON if: SYSCLK = OSC_CLK
  - PLL0ON, PLL1ON should be ON if: SYSCLK = PLL0_CLK or PLL1_CLK
  - Configuration “00” for the FLAON bit fields should not be used
  - System clock configurations marked as ’reserved’ may not be selected
  - Configuration "1111" for the SYSCLK bit field is allowed only for TEST mode
- Preventing Invalid Mode Transition
  - Mode requested when a transition is active (mode transition illegal)
  - Target mode not valid with respect to the current (mode request illegal)
  - Target mode is disabled in mode enable register (disable mode access)
  - Target mode doesn’t exist (non existing mode access)

5 HALT low power mode basic features

This mode is intended as a top-level low power mode where the platform is shut down but the system clock can remain the same as the run mode.This is a low power mode with reduced activity during which the core clock is disabled. It can be configured to switch off analog peripherals like PLL, Flash, Main Voltage Regulator etc. for efficient power management at the cost of higher wake-up latency. Software request to HALT mode can be triggered from RUN0-3 and not from DRUN.

6 STOP low power mode basic features

This mode is intended as an advanced low power mode during which the clock to the platform is stopped. It can be configured to stop the clock of most peripherals, including the oscillator, for efficient power management at the cost of increased wake-up latency. Comparing to the HALT mode PLL can’t be enabled and additionally I/O Output Power Down Control (PDO) allows to switch off pad power sequence driver while the state of the output remains functional. Software request to STOP mode can be triggered from RUN0-3 and not from DRUN.

7 STANDBY low power mode basic features

This chapter provides details and focuses STANDBY low power mode as this mode differs in handling and behavior comparing to the rest of the available modes.

STANDBY mode is intended:

- As an extreme power saving mode with everything turned off except the circuits needed to allow the device to wake-up
- To be used by software to remain in the lowest power consumption state with no requirement to wake up quickly

This is a reduced leakage low power mode in which only PD0 power domain (MC_RGM, MC_PCU, WKPU, 8K RAM, RTC_API, SSWU, ADC_STDBY, SIRC, FIRC, 32 kHz XTAL, SSCM, and VREG) is connected and power supply is cut off from most of the device. Optionally power domains PD2 and PD3 could be configured extending STANDBY RAM up to 256K. Please refer to Device power management framework in *Section 7.4*.

All SPC58xB/C/G/H derivatives provide 8 K STANDBY RAM by default. It could be extended by configuration up to maximum 256 K of STANDBY RAM on SPC58xGx. Please refer to: STANDBY RAM Size Configurations.

Wake-up from this mode takes a relatively long time, and content is lost and must be restored from backup.

The output sequence of this mode is similar to the reset sequence and it is a major difference from other low-power modes such as HALT or STOP.
We also need to reconfigure all modules after the power has been interrupted in STANDBY mode.

In addition to the booting from the default Flash location after STANDBY exit, the microcontroller can also be configured to boot from the STANDBY (backup) RAM.

7.1 STANDBY mode wake-up in backup RAM

This feature supports the implementation of periodical STANDBY application waking up in RAM executing special sleep application while FLASH is still kept in power down mode. It allows to decrease current consumption as much as possible and significantly shorten wake-up time.

Device (Master IO Core = Core 2) boots from the address specified in ME_CADDR0 register. In case that boot from backup RAM is requested, ME_CADDR0 should address STANDBY RAM. (STANDBY RAM start address = 0x400A8000)

Before starting the STANDBY mode sequence an application has to copy special sleep application in STANDBY RAM and configure ME_CADDR0 appropriately.

7.2 STANDBY mode and power supply

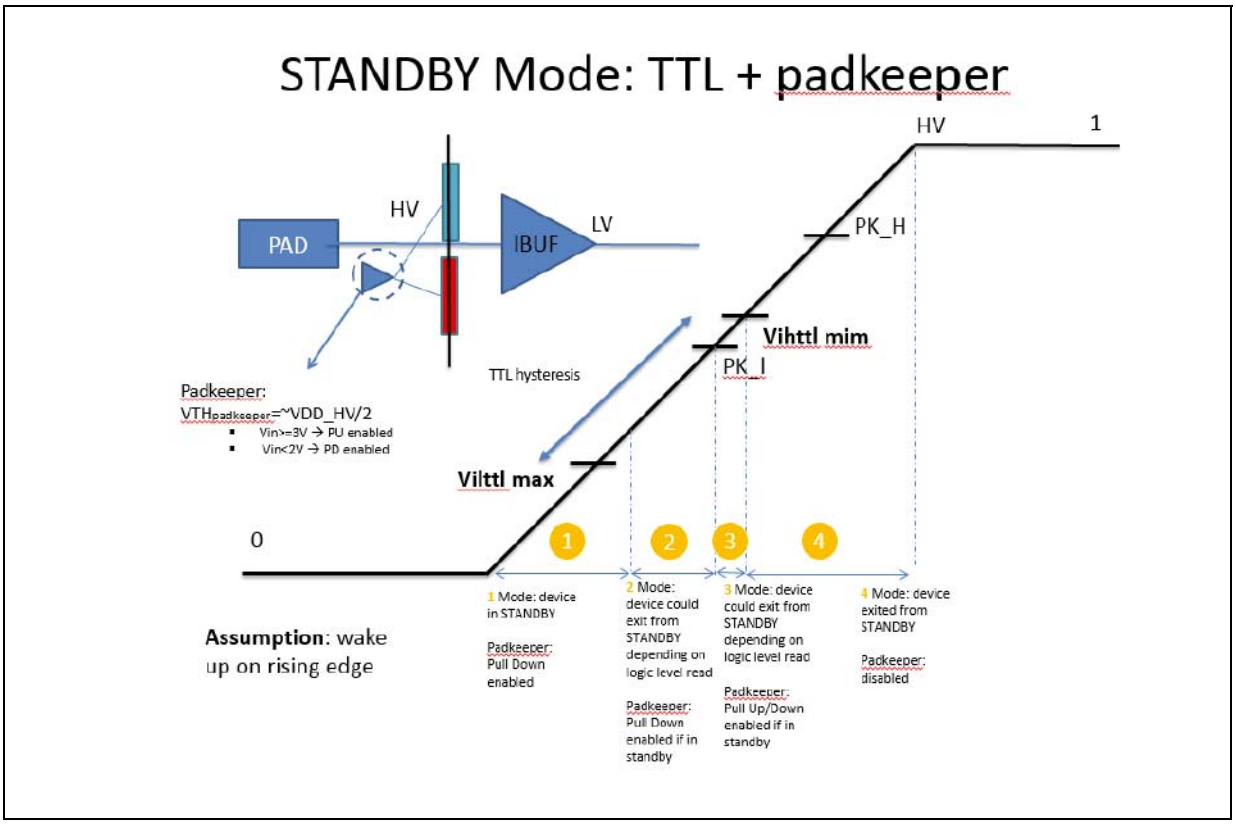
STANDBY mode is not supported when the device is operating with External regulator or SMPS regulator mode. STANDBY mode is supported only in devices configured with internal or external ballast transistor.

7.3 MCU pads in STANDBY mode

There are two types of pads during STANDBY mode:

- Not Low Power Pads
  These pads are not active in STANDBY mode and when STANDBY is entered, both input and output buffers inside the pins are disabled. Also internal pull-up / down are disabled.
  Floating state will not affect the power consumption because of isolation within the pad is automatically applied.
- Low Power Pads
  These pins are active during STANDBY mode and when the device enters into STANDBY mode, these input pads are automatically configured in TTL mode. It is recommended to configure wake-up pads in TTL mode in RUN modes as well.
  Moreover when the device enters the STANDBY mode, the Pad-Keeper feature is activated for inputs:
  If the pad voltage level is above the pad keeper high threshold, a weak pull-up resistor is automatically enabled.
  If the pad voltage level is below the pad keeper low threshold, a weak pull-down resistor is automatically enabled.

Figure 3. STANDBY mode pad keeper



7.3.1 LP pads with OPC pad latching:

In devices embedding SSWU module, some low power pads (see device excel pinout, sheet ‘IO signal table’, column ‘Function’) when their output buffer is enabled before

entering in STANDBY mode, it can be configured for having a predefined value exiting from STANDBY itself. OPC latching can be individually enabled / disabled via SIUL2.SCR0 register, field PADxx_MUX_SEL.
Default configuration is OPC latching enabled.
Field PADxx_OPC_MASK allows to enable / disable OPC driving during STANDBY mode.
By default OPC channel drives pad during STANDBY mode.

Table 4. LP pads keeping value after STANDBY exit

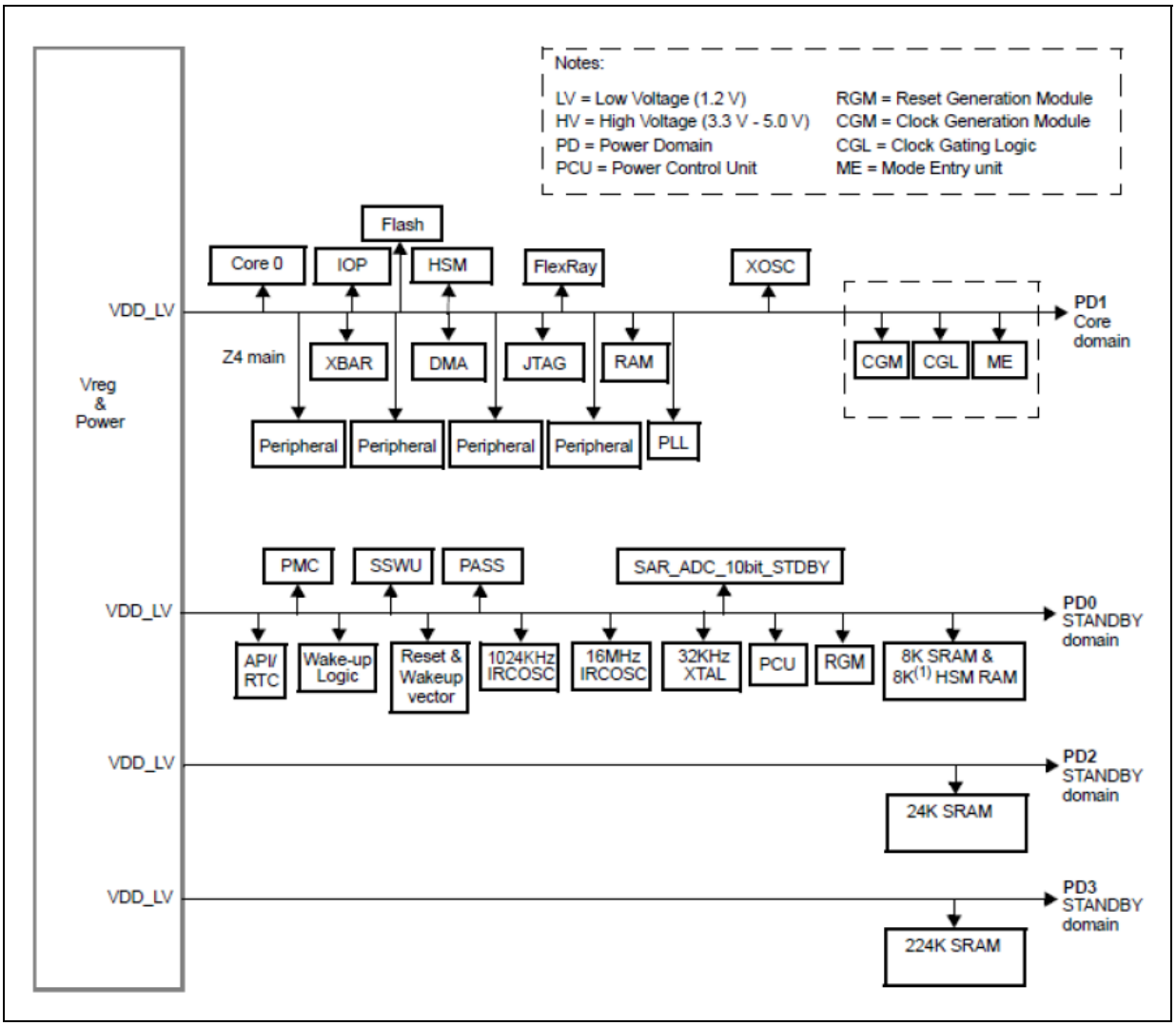
Device	LP Pads with OPC latching configurable via SIUL2.SCR0.PADxx_MUX_SEL
SPC582Bx	PB[8], PB[9], PB[10]
SPC584Bx	PB[8], PB[9], PB[10],PD[5], PF[9], PF[10], PF[11], PF[12]
SPC58xCx	PB[8], PB[9], PD[5], PF[9], PF[10], PF[11], PF[12]
SPC58xGx	PB[11], PD[10], PF[3], PF[4], PF[9], PF[12], PF[15], PG[15]
SPC58xHx	PB[8], PB[9], PD[5],PF[9], PF[10], PF[11], PF[12], PH[0],PH[1] PH[2], PH[3]

Table 5. MCU pads status during STANDBY mode

Pad Type	Before STANDBY	During STANDBY	After STANDBY
Non LP Pads	Any	OFF (HighZ)	Back to reset configuration
LP Pads	Any	Input buffer enabled + Pad Keeper	Back to reset configuration
LP Pads with OPC	Input	Input buffer enabled + Pad Keeper	Back to reset configuration
	Output (Value)	OPC enabled	Keep Value

7.4 STANDBY mode and device power management framework

Figure 4. Device power management framework



1. Enabled during stand-by depending on a DCF bit.

MCU provides 4 power domains.

Only PD2 and PD3 are configurable by the user. (PCU_PCONF2, PCU_PCONF3).

These power domains allow to extend STANDBY RAM up to 256 kB.

7.5 STANDBY mode application considerations

The following several normal behaviors and properties have to be considered by the application programmer utilizing STANDBY mode to gain minimal current consumption:

- All wake-up lines have to apply pull up resistors during STANDBY mode regardless if resistor is internal or external, otherwise current leakage during STANDBY mode occurs. (in worst case 1.9 mA per pad, typically 100 µA per pad) Other remaining I/O pads are not power supplied during STANDBY mode, they are in high impedance state

and therefore it is not possible keeping high or low logic level on these pads during STANDBY mode. Pad Keeper functionality applies pull up/down internal resistors by default and therefore configuring internal pull up resistors via WKUP.WIPUER.R is not necessary any more. (Previous Body MCUs family does not provide Pad Keeper functionality)

- If there is a STANDBY mode request while any wake-up event is active (wake-up flag is set within WKUP.WISR.R), the microcontroller will not enter STANDBY mode and code execution continues after STANDBY mode request command. It means all wake-up events have to be handled and appropriate flags have to be cleared before STANDBY mode entering.
- Additional following particular configuration values within configuration register ME_DRUN_MC are retained through STANDBY mode: FLAON and FXOSCON. User can speed up starting FXOSC and FLASH immediately during STANDBY exit.
- It is illegal to switch the Flash directly from low-power mode to power-down mode or from power-down mode to low-power mode. The MC_ME, however, does not prevent this nor does it flag it.
- After MCU POR Clock Monitor Unit 0 parameter RCDIV is set to CMU_0.CSR.B.RCDIV = 3, but it is cleared during STANDBY mode. For XTAL<=16 MHz has to be set again after wake-up from STANDBY otherwise next mode transition into run mode using XTAL can stall.

7.6 STANDBY mode entering considerations

This chapter provides only internal and specific hardware information concerning Mode Entry module needed to better understand the behavior during transition into STANDBY mode in particular configuration sequence which can cause issues. Basic information about Module Entry module is available inside RM.

Concerning STANDBY mode, RM provides important notes and warnings. The following warning is highlighted inside RM:

The MC_ME automatically requests peripherals to enter their stop modes if the power domains in which they are residing are to be turned off because of mode change. However, it is good practice for software to ensure that those peripherals which are to be powered down are configured in the MC_ME to be frozen (see reference manual SPC58xCx rev2 chapter 58.4.3.3 “Peripheral clocks disable”).

We can say that Auxiliary Clock Dividers also need to be kept enabled (if previously enabled) before attempting STANDBY mode entry transition.

In general, it is not recommended to disable these Auxiliary Clock Dividers while managing clock gating to the peripherals through MC_ME. Whenever a peripheral is requested to be frozen, there is special handshake executed between MC_ME and the peripheral prior to gating the clock. In this handshake, a stop request is sent to peripheral and ME waits for an acknowledgment before proceeding to further steps of mode transition. While executing this handshake, clock to the peripheral must be present.

7.6.1 Recommended configuration preventing issue

Disabling auxiliary clock dividers is not considered an effective action to prevent the issue. The suggestion is to use PCTL / PC configuration registers for particular module clock gating when needed, instead of disabling Auxiliary Clock Dividers.

Next recommendation is to ensure that those peripherals, that should be powered down in STANDBY mode are configured in the MC_ME to be frozen (module clock is gated by PCTL/PC registers).

7.6.2 Gating clock of running peripheral

Before peripheral gating clock, it is required to stop peripheral which is configured to trigger interrupt. Otherwise consequent mode transition can stall. It affects all mode transitions, not only transition in *STANDBY* mode.

In other words all running peripherals (e.g. timers triggering interrupts) should be properly stopped before gating clock to them via PCTL / PC registers.

7.6.3 Safe sequence for STANDBY mode entering

1. Stop all peripherals
2. Handle all pending interrupts (e.g. for DSPI, all data have to be transmitted out and received data read out from receive FIFO and related flags TCF and RFDF cleared)
3. Disable interrupts
4. Disable clock for all clocked peripherals via PCTL and move to DRUN with FIRC clock
5. Handle and clear all wake-up flags (WKPU.WISR)
6. Enter STANDBY mode

Recommended sequence after wake-up from *STANDBY* mode:

1. After wake-up inspect RGM.FES and DES registers (if any reset or POR event occurred)
2. According to previous status inspect FCCU status registers
3. Inspect wake-up flags WKPU.WISR (to know which source triggered wake-up)
4. Handle flagged events and clear flags before new STANDBY period

7.7 RTC / API - Real time clock/autonomous periodic interrupt

It is one 32-bit free running timer available during STANDBY mode and allows to generate internal wake-up event. It runs in all modes of operation, including normal RESET.

3 selectable counter clock sources in STANDBY mode:

- SIRC (128 kHz)
- SXOSC (32 kHz) (pads for external SXTAL connection are not available on all packages)
- IRCOSC (16 MHz)

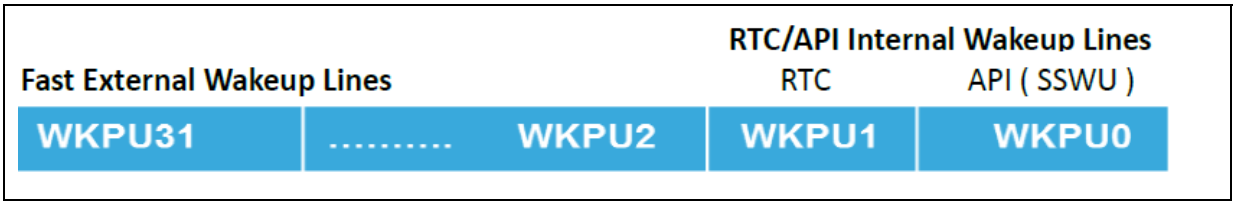
Periphery provides configurable wake-up event for RTC match, API match (SSWU), and RTC rollover. API allows autonomous periodic wake-up generation. It means next offset is recalculated by hw automatically. RTC comparator needs to recalculate next offset by software.

2 compare registers defining wake-up period:

- APIVAL - API 32 bit compare value defines period for API wake-up source
- RTCVAL - RTC 32 bit compare value defines period for RTC wake-up source

7.8 WKPU wake-up lines

MCU provides two internal wake-up lines which are mapped as follows:



SSWU is not available on SPC582Bx.

7.9 STANDBY low power clocks

7.9.1 SIRC – slow 128 kHz RC oscillator

In all modes except STANDBY0, Low Power RC oscillator (LPRC) should be enabled by PMC digital interface clearing MISC_CTRL_REG[RCOSC1M_ENB] bit[a].

In STANDBY0 mode instead, Low Power RC oscillator is always enabled.

Note: SIRC (128 kHz) can't be trimmed.

7.9.2 SXTAL – slow 32 kHz crystal oscillator

SXTAL is not available on SPC58xBx and not on all packages and not populated on ST SPC58 EVBs. On SPC58xC/G it can be enabled in the OSC32K_DIG.CTL.R register:

OSC32K_DIG.CTL.B.OSCON = 1

7.9.3 IRC – fast 16 MHz RC oscillator

16MHz Fast IRC can be enabled in the MC_ME.STANDBY0_MC.R register:

MC_ME.STANDBY0_MC.B.IRCON = 1

7.10 SSWU - Smart Standby Wake-up unit

SSWU is not available on SPC582Bx and RTC/API wake-up channel WKPU0 is available for user. Otherwise SSWU is linked to the RTC / API wake-up channel.

The Smart Standby Wake-up unit (hereafter referred as SSWU) is aimed at reducing further the device power consumption by scheduling, on a RTC time basis, a sequence of user-programmable tasks, which are executed without the CPU intervention and which are intended to determine whether the whole MCU shall be waken-up or not, based on user-defined digital and / or analog measurements. In addition, it shall be possible to schedule a system wake-up based on a user defined time interval.

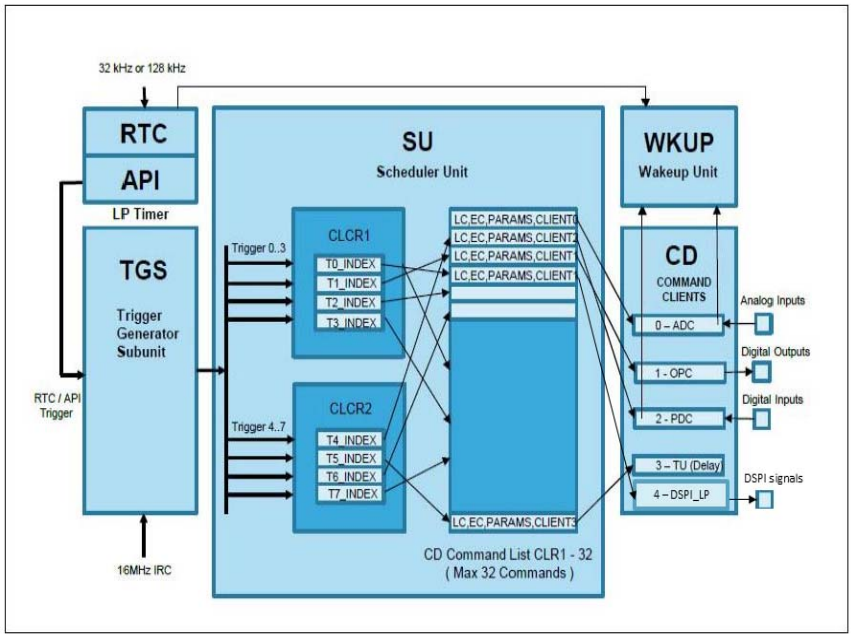
a. On SPC584Bx and SPC58NHx devices the LPRC is enabled during power on reset sequence and it is not expected that application switches off the LPRC oscillator.

SSWU consists of the following blocks:

- RTC = Real Time Clock
- Stand-by eCTU = Stand-by Enhanced Cross Triggering Unit
- CD = Command Decoder
- ADC = Analog Digital Converter
- PDC = Port Digital Comparator
- OPC = Output Pin Control block
- TU = Timer Unit
- DSPI_LP = Low Power DSPI (client) (only for SPC58xHx)
- PMC1 = Peripheral Control Module for DSPI_LP frames (only for SPC58xHx)

See *Section 7.13.4* for an example about blocks setup during SSWU initialization.

Figure 5. STANDBY SSWU diagram



The SSWU IPs is clocked with the internal 16 MHz IRC oscillator.

Even if IRCOSC is configured off in STANDBY through the ME_STANDBY0_MC register, it will be automatically turned on once SSWU is activated. The IRCOSC will be automatically turned off after SSWU Stop command is executed. The ADC IP runs at 8 MHz (IRCOSC clock divided by 2).

7.10.1 ADC

The ADC block is a reduced, low-power version of the Successive Approximation Register Analog-to-Digital Converter called 10-bit SARADC_STDBY. Its main functionality is to

compare the converted value with a watchdog, and trigger a wake-up event in case the value is out of the intended range. The watchdog thresholds (low and upper limits) are linked (that is pre-programmed during the RUN mode) to each ADC channel, so that a specific channel has its own threshold limit. The ADC configuration comprises:

- 24 ADC internal channels (maximum amount, depends on particular derivative and package)
- 8 ADC external channels
- 4 different configurable Analog Watchdog threshold values

7.10.2 OPC

The OPC block is used to drive the digital output pin to a desired level. The intended functionality is to open / close external switches. The Stand-by eCTU command parameter contains the port pin and level.

There is a possibility to configure OPC pads behavior in order to keep the previous value.

Soc Configuration Register 0 (SIUL2_SCR0) allows to configure behaviors as follows:

1. Upon entering STANDBY mode, the OPC pad values can be maintained via SIUL2_SCR0.PADx_OPC_MASK
   By default OPC pad is driven during the STANDBY mode.
2. Upon exiting from STANDBY mode, the OPC pad values can be maintained via SIUL2_SCR0.PADx_MUX_SEL
   By default pad value is maintained.
3. If the pad is used for OPC functionality during STANDBY mode, on the subsequent STANDBY mode entry the PAD value is restored to the value during the previous STANDBY mode exit.

7.10.3 PDC

The PDC block is targeted to measure the level of a set of pins (up to 16) and to generate a system wake-up event when the level is acknowledged at a specified level and, optionally, for a certain time. Two commands are needed for the PDC:

- CMP 0 / CMP 1: to compare a pin level versus a 0 or 1
- CHECKOUT: to trigger a wake-up in case the result of the CMP commands for that specific pin executed from the beginning of the Smart Wake-up Sequence is always ok.For each pin, the PDC shall latch the comparison results, which shall report whether the target value was always matched (whenever a PDC.CMP0/1 command is launched) or not. When checkout command is launched for a specific pin, it will trigger a wake-up event only if the target level was steadily acknowledged along all the comparisons done for that pin.
  Useful for contact monitoring within Sleep Mode Applications.

7.10.4 TU

The TU block is used to create a programmable DELAY between the command execution. It needs only one command with the DELAY value programmed as a parameter. With clock period = 62.5 ns and WAIT value settable up to 128, the DELAY is programmable up to 8 µs.

Further it could be applied SSWU PRESCALER (up to divider 128) and prolong max delay value up to 1.016 msec.

PMCDIG.SSWU_CTRL_REG.B.SSWU_PRESCALER = 0; (SSWU PRESCALER = 1)

Time delay = DELAY * $T_{RC16MHz}$* TU_{PRESC}

In case the TU parameter (DELAY time) is set to 0, this is decoded as a command request to stop the whole SSWU sequence.

The two commands which can cause wake-up (ADC-ADC or ADC-PDC or PDC-PDC) shall be not executed one after another without inserting a TU delay command.

7.10.5 DSPI frame

The SSWU engine is able to trigger the delivery of a DSPI frame. Once the DSPI client is selected by a command, the SSWU will activate a trigger to transmit a DSPI frame. The DSPI frame value is set by the content of the PCM_1 register pointed by the PARAMS field value of the SSWU command. The transfer of the value from the PCM_1 module to the DSPI_LP module is performed by using the DSPI DSI bus.

The DSPI_LP is not requested to process message reception during the stand-by mode.

The DSPI following characteristics can be preset via software in the DSPI register interface before entering the stand-by mode and cannot be changed during the stand-by phase:

- Baud-rate (up to 4 Mbps)
- Frame size: 4, 8, or 16-bit
- Polarity
- Phase

The DSPI_LP module can be fully working like a regular DSPI when in non-stand-by mode. DSPI_LP is needed to be programmed in Hardware Trigger (HT) Mode during SSWU operation.

7.10.6 SSWU parameters

Starting to execute the first command, as programmed in the pointer associated to a specific SU trigger, each command is executed after the previous one is completed. The "LC" command flag determines whether the current command is the last one to be executed for the sequence of commands being executed.

The *Table 6* shows the maximum number of the available SSWU/wake-up lines. For smaller MCU packages it can be fewer and has to be aligned with the IO definition file (SPC584Bx and SPC58xCx – OPC0 is missing) for particular selected MCU derivative and package.

Table 6. SSWU parameters (wake-up, SSWU ADC, OPC, PDC lines)

Device	ADC CHNLs	ADC Ext CHNLs(1)	ADC WDGs (2)	OPC	PDC	WKPU EXT	WKPU INT
SPC582Bx	NA	NA	NA	NA	NA	24	API, RTC
SPC584Bx	24	8	4	7	14	24	SSWU, RTC
SPC58xCx	24	8	4	7	29	28	SSWU,RTC
SPC58xGx	8	8	4	8	4	30	SSWU, RTC
SPC58xHx	24	8	4	11	31	54	SSWU, RTC

1. ADC External channels require analog multiplexer selection signals (via OPC).
2. ADC Watchdog different threshold levels.

7.11 HSM in STANDBY mode

HSM dedicated RAM (40 KB) kept during STANDBY mode can be 8 kB when appropriately configured via security DCF record. HSM_EN_STANDBY option keeps 8 kB of HSM RAM during STANDBY mode and allows HSM to boot from its RAM (ME_CADDR4) after STANDBY mode exit (wake-up).

7.12 STANDBY mode and software watchdog timer

It is not possible to run MCU internal SWT during STANDBY mode. Therefore it is necessary to use the external one. Typically external Software Watchdog Timer is located in so called SBC (System Basis Chip) and refreshed via SPI during periodical wake-up period. This approach is used by periodical low power applications described in the next chapter.

7.13 Periodical low power applications

Periodical Low Power Application refers to special “Sleep Mode” applications where minimal consumption is paramount. It is applied e.g. in car central body module during parking while power supply is never switched off. Some limited functionality is still required.

The approach allows to reach the lowest average consumption approx. 100 µA @25 °C providing contact monitoring via SSWU on SPC584Bx.

External watchdog refresh via SPI or Real Time Clock management (temperature compensation) can be performed only via wake-up into Sleep RAM based special application.

7.13.1 Contact monitoring

A contact monitoring is intended the feature to periodically determine contact state changes, wake-ups and switches to main application when an external event happens. Time base for periodical timing is usually RTC / API and contacts could be digital or analog. Even sensors connected via LIN or SPI can be monitored. External watchdog refresh connected via SPI is usually managed as well.

A second type of contacts (wake-up contacts) are connected directly to MCU’s specially designed WKPU inputs which allow immediate wake-up from STANDBY mode. These are usually used for fail-safe input signals and wake-up signal from SBC (System Basic Chip). SBC usually contains CAN and LIN transceivers and determines immediate wake-up from STANDBY mode via CAN or LIN wake-up message.

7.13.2 Sleep mode application

So called small compact Sleep Mode Application runs out of the first part of RAM which is backup during STANDBY low power mode and while flash is still kept in low power down mode and system clock is FIRC 16 MHz.

In addition to the booting from the default location after STANDBY exit, the microcontroller can also be configured to boot from the backup RAM. Register ME_CADDR0 allows to configure boot starting address (backup RAM start address = 0x400A8000).

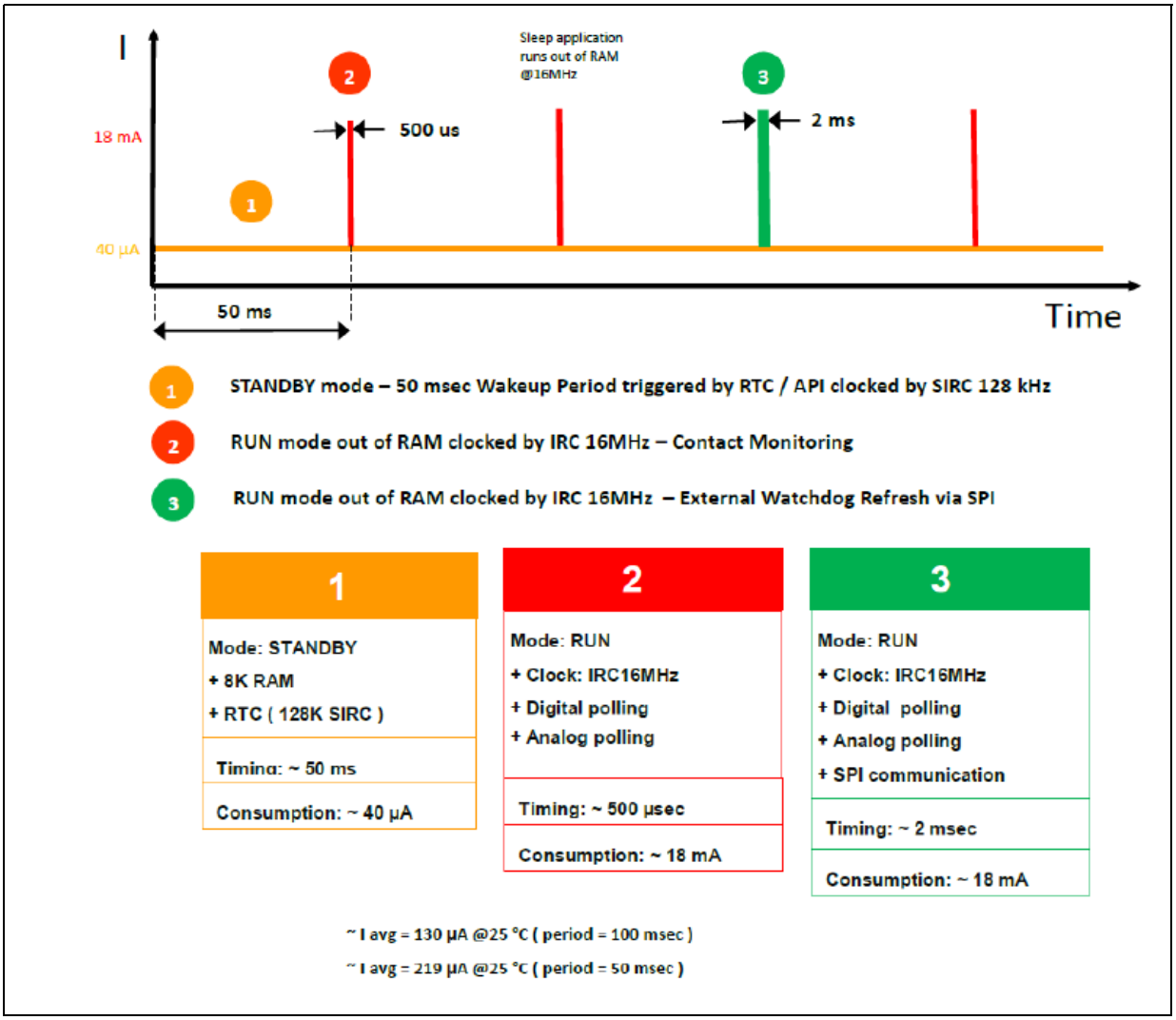
Examples following in this chapter keep the default PCONF[2] and PCONF[3] configuration. It means the during the RUN mode the whole RAM is ON and included in RUN consumption values.

User can additionally optimize RUN mode via mentioned registers allowing to keep ON only 8KB.

During STANDBY mode only default 8KB RAM is ON. If user enables e.g. for SPC582B 64KB RAM in STANDBY mode then current consumption increases typically about 15 uA.

If user enables for SPC58xH 256KB RAM in STANDBY mode then current consumption increases about 60 uA.

Figure 6. Sleep mode application diagram

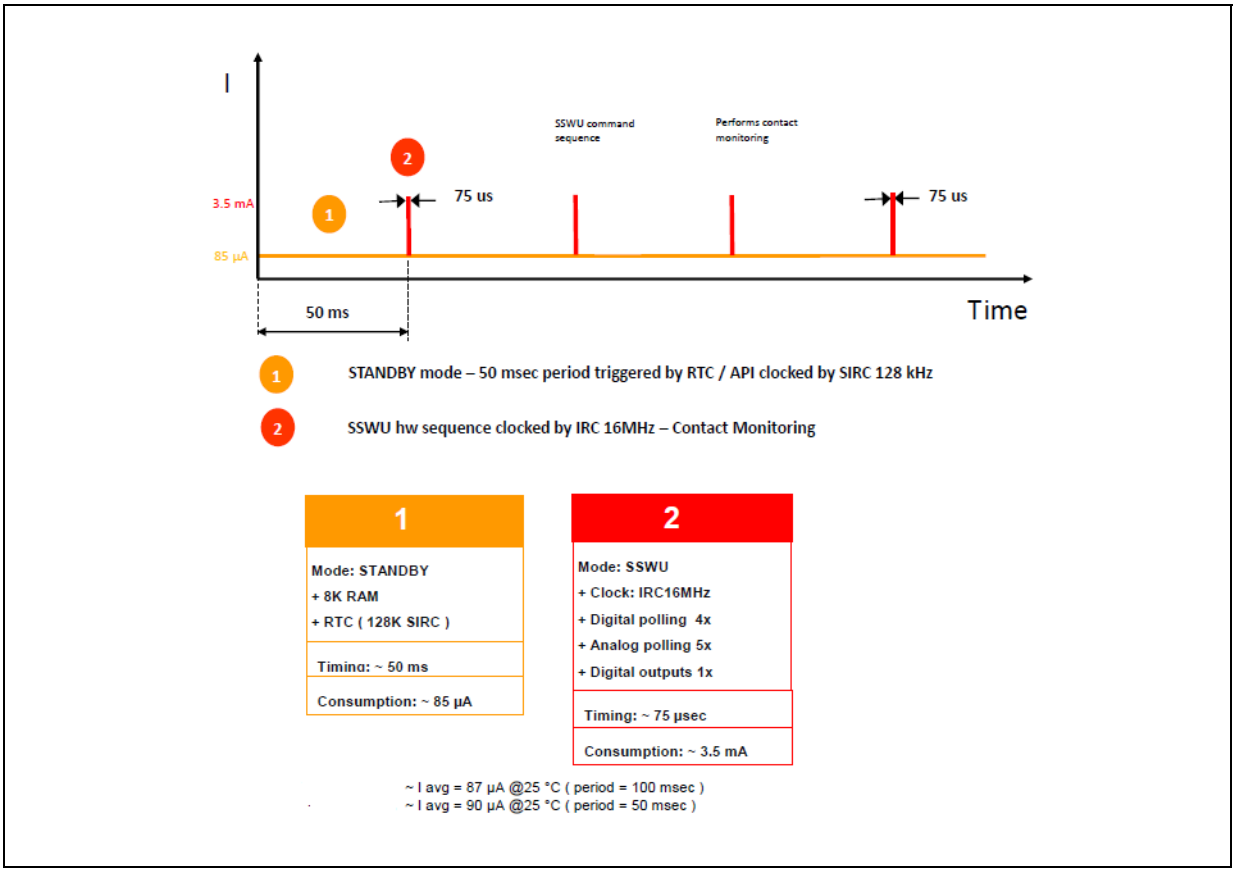


7.13.3 SSWU utilized in sleep mode application

Smart Standby wake-up Unit (STANDBY eCTU) is convenient to perform Periodical Sleep Mode application where there is no requirement for more complex computational task as temperature compensation, SPI, LIN communication etc. Then wake up and run special

RAM sleep application are not necessary at all and sequence is periodically performed without any CPU intervention.

Figure 7. SSWU sleep mode sequence diagram



7.13.4 SSWU contact monitoring application example

This is an example application performing contact monitoring each 50 msec via SSWU on SPC58xGx cut 2.1. Particular Errata PS1816 present on cut 2.1 has to be taken into account. For SPC58xCx/SPC584Bx additionally Errata PS2370 has to be considered as well.

Application shows how to configure SSWU on SPC58xGx to perform the following contact monitoring sequence:

1. Periodical contact monitoring each 50 msec
2. 5x ANALOG INPUTS
3. 4x DIGITAL INPUTS
4. 1x DIGITAL OUTPUT

7.13.4.1 SSWU initialization code sequence:

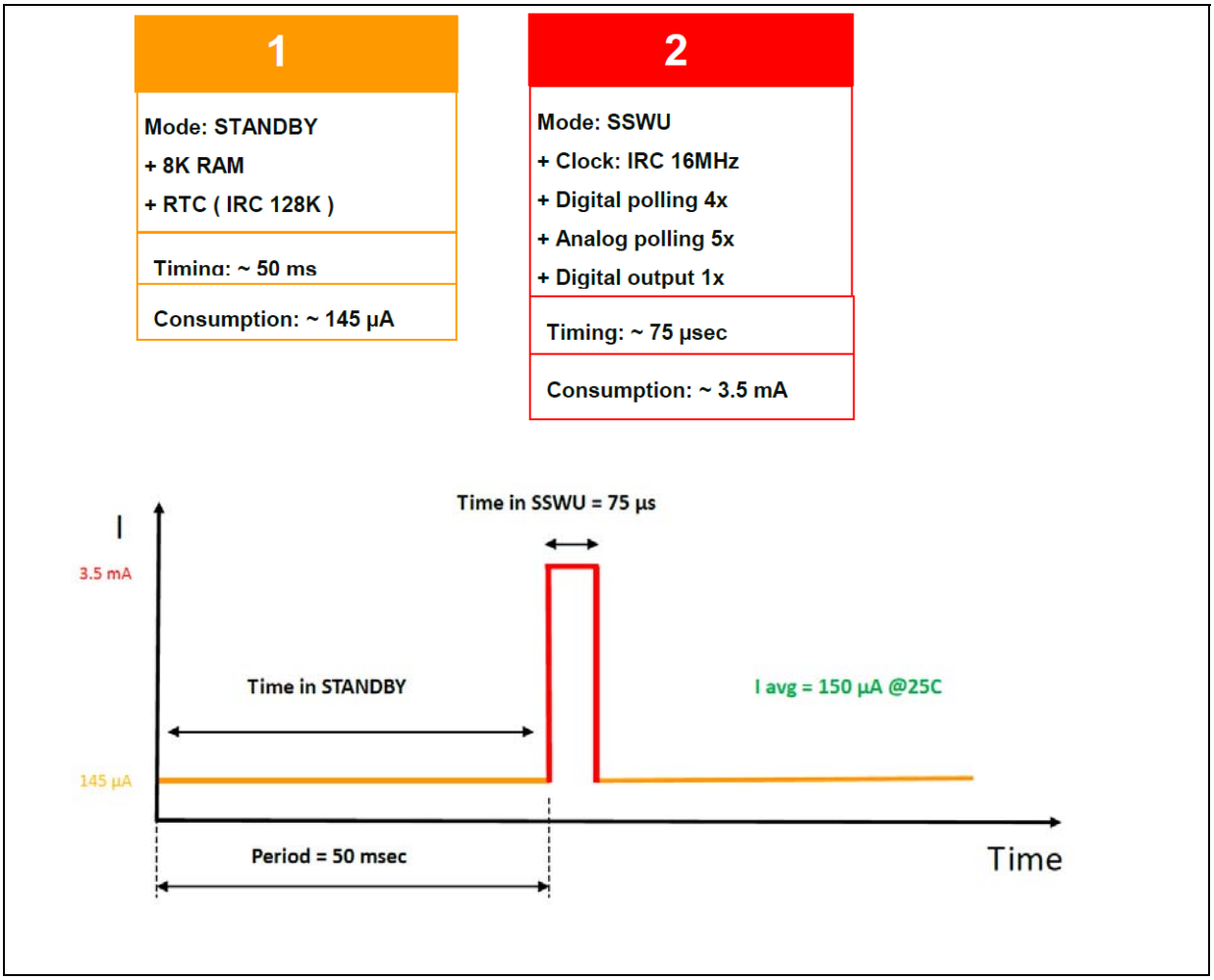
```
/*****************************************************************
* FUNCTION      : SSWU_init
*
* DESCRIPTION   : Inits SSWU
*
*
* INPUTS        :
*       - None
*
* OUTPUTS
*       - None
*****************************************************************/
void SSWU_init() {

    //-------------------------------------------------------------------------
    //  5x ANALOG INPUTS + 4x DIGITAL INPUTS + 1x DIGITAL OUTPUT
    //  CH0 AN88, CH1 AN89, CH2 AN90, CH3 AN91, CH4 AN92  ( ANALOG INPUTS )
    //  PDC0, PDC1, PDC2, PDC3 ( DIGITAL INPUTS )
    //  OPC0 ( DIGITAL OUTPUT )
    //-------------------------------------------------------------------------

    //Configure SSWU clock prescaler => div 1
    PMCDIG.SSWU_CTRL_REG.B.SSWU_PRESCALER = 0;

    //----------------COMMAND BLOCK START SIGNAL ( 1x DIGITAL OUTPUT ) ---------

    //First command list
    STDBY_CTU_0.CLR[0].R          = 0x20000801;        //set OPC0 pin to '1'
```

```
//------------------------- 5x ANALOG INPUTS -----------------------------

STDBY_CTU_0.CLR[1].R          = 0x20000000;        //convert ADC_STANDBY_CH0 (AN88)
STDBY_CTU_0.CLR[2].R          = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[3].R          = 0x20000100;        //convert ADC_STANDBY_CH1 (AN89)
STDBY_CTU_0.CLR[4].R          = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[5].R          = 0x20000200;        //convert ADC_STANDBY_CH2 (AN90)
STDBY_CTU_0.CLR[6].R          = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[7].R          = 0x20000300;        //convert ADC_STANDBY_CH3 (AN91)
STDBY_CTU_0.CLR[8].R          = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[9].R          = 0x20000400;        //convert ADC_STANDBY_CH4 (AN92)
STDBY_CTU_0.CLR[10].R         = 0x20003F03;        //set delay to 63 => 4 usec

//------------------------- 4x DIGITAL INPUTS -----------------------------

STDBY_CTU_0.CLR[11].R         = 0x20002002;        //test PDC0 input to '1'
STDBY_CTU_0.CLR[12].R         = 0x20003F03;        //set delay to 63 => 4 usec
STDBY_CTU_0.CLR[13].R         = 0x20004002;        //CHECKOUT PDC0 input
STDBY_CTU_0.CLR[14].R         = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[15].R         = 0x20002102;        //test PDC1 input to '1'
STDBY_CTU_0.CLR[16].R         = 0x20003F03;        //set delay to 63 => 4 usec
STDBY_CTU_0.CLR[17].R         = 0x20004102;        //CHECKOUT PDC1 input
STDBY_CTU_0.CLR[18].R         = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[19].R         = 0x20002202;        //test PDC2 input to '1'
STDBY_CTU_0.CLR[20].R         = 0x20003F03;        //set delay to 63 => 4 usec
STDBY_CTU_0.CLR[21].R         = 0x20004202;        //CHECKOUT PDC2 input
STDBY_CTU_0.CLR[22].R         = 0x20003F03;        //set delay to 63 => 4 usec

STDBY_CTU_0.CLR[23].R         = 0x20002302;        //test PDC3 input to '1'
STDBY_CTU_0.CLR[24].R         = 0x20003F03;        //set delay to 63 => 4 usec
STDBY_CTU_0.CLR[25].R         = 0x20004302;        //CHECKOUT PDC2 input
STDBY_CTU_0.CLR[26].R         = 0x20003F03;        //set delay to 63 => 4 usec

//------------- COMMAND BLOCK END SIGNAL ( 1x DIGITAL OUTPUT ) -------------

STDBY_CTU_0.CLR[27].R         = 0x20000001;        //set OPC0 pin to '0'

STDBY_CTU_0.CLR[28].R         = 0x60000003;        //add a delay of 0 usec to stop the sequence

//------------------------------------------------------------------------

STDBY_CTU_0.CLCR1.B.T0_INDEX  = 0;                 //trigger0 points to CLR[0]
STDBY_CTU_0.THCR1.R           = 0x00000011;        //trigger0 enable

STDBY_CTU_0.TCR[0].R          = 0x00000001;         //trigger0 generated quite immediately
```

```
STDBY_CTU_0.TGSCCR.R          = 0x500;        //SSWU turned off after 80 us
 STDBY_CTU_0.TGSISR.R          = 1;           //ctu_trg_in[0] (API) Rising edge Enable
 STDBY_CTU_0.CTUCR.B.TGSISR_RE = 1;           //TGS Input Selection Register Reload Enable
 STDBY_CTU_0.CTUCR.B.GRE       = 1;           //General Reload Enable

 STDBY_CTU_0.CTUCR.B.CTU_FSM_RESET = 1;     //PS1816 Errata => reset CTU FSM before next stand-by entry

 //-------------------------------------------------------------------------
 // SAR_ADC_10bit_STDBY configuration
 //-------------------------------------------------------------------------
 // CH88 PB[8] / CH89 PB[9] / CH90 PB[10] / CH91 PB[11] / CH92 PD[9]
 //-------------------------------------------------------------------------

 SAR_ADC_10BIT_STDBY.MCR.B.OWREN         = 1;     //SAR ADC 0 overwrite enable set
 SAR_ADC_10BIT_STDBY.MCR.B.MODE          = 0;     //Normal Mode
 SAR_ADC_10BIT_STDBY.MCR.B.CTU_MODE      = 1;     //Trigger mode is selected
 SAR_ADC_10BIT_STDBY.MCR.B.CTUEN         = 1;     //CTU enabled
 SAR_ADC_10BIT_STDBY.MCR.B.PWDN          = 0;     //Power up SAR ADC 0

 SAR_ADC_10BIT_STDBY.ICNCMR2.B.NCE_CH88 = 1; //sets channels 88 to be converted in SARADC_10bit_standby
 SAR_ADC_10BIT_STDBY.ICNCMR2.B.NCE_CH89 = 1; //sets channels 89 to be converted in SARADC_10bit_standby
 SAR_ADC_10BIT_STDBY.ICNCMR2.B.NCE_CH90 = 1; //sets channels 90 to be converted in SARADC_10bit_standby
 SAR_ADC_10BIT_STDBY.ICNCMR2.B.NCE_CH91 = 1; //sets channels 91 to be converted in SARADC_10bit_standby
 SAR_ADC_10BIT_STDBY.ICNCMR2.B.NCE_CH92 = 1; //sets channels 92 to be converted in SARADC_10bit_standby

 //-------------------------------------------------------------------------
 // SAR_ADC_10bit_STDBY => 4 ANALOG WATCHDOGS
 //-------------------------------------------------------------------------

 SAR_ADC_10BIT_STDBY.WTHRHLR0.B.THRH       = 0x333; //select high THRHLR0 threshold = 4.0 V
 SAR_ADC_10BIT_STDBY.WTHRHLR1.B.THRH       = 0x2E1; //select high THRHLR1 threshold = 3.6 V
 SAR_ADC_10BIT_STDBY.WTHRHLR2.B.THRH       = 0x2CD; //select high THRHLR2 threshold = 3.5 V
 SAR_ADC_10BIT_STDBY.WTHRHLR3.B.THRH       = 0x2B9; //select high THRHLR3 threshold = 3.4 V

 SAR_ADC_10BIT_STDBY.WTHRHLR0.B.THRL       = 0;     //select low THRHLR0 threshold = 0 V
 SAR_ADC_10BIT_STDBY.WTHRHLR1.B.THRL       = 0;     //select low THRHLR1 threshold = 0 V
 SAR_ADC_10BIT_STDBY.WTHRHLR2.B.THRL       = 0;     //select low THRHLR2 threshold = 0 V
 SAR_ADC_10BIT_STDBY.WTHRHLR3.B.THRL       = 0;     //select low THRHLR3 threshold = 0 V

 //-------------------------------------------------------------------------
 // 5 ANALOG INPUTS
 //-------------------------------------------------------------------------

 //ADC_STANDBY_CH0 = Internal Channel 88;
 SAR_ADC_10BIT_STDBY.ICWSELR11.B.WSEL_CH88 = 0;    //select THRHLR0 treshold for CH88
 SAR_ADC_10BIT_STDBY.ICWENR2.B.WEN_CH88       = 1;   //enable watchdog for CH88
 SAR_ADC_10BIT_STDBY.WTIMR.B.MSKWDG0H          = 1;   //enable high THRHLR0 threshold interrupt

 //ADC_STANDBY_CH1 = Internal Channel 89;
 SAR_ADC_10BIT_STDBY.ICWSELR11.B.WSEL_CH89 = 1;   //select THRHLR1 treshold for CH89

 SAR_ADC_10BIT_STDBY.ICWENR2.B.WEN_CH89      = 1;   //enable watchdog for CH89
```

```
SAR_ADC_10BIT_STDBY.WTIMR.B.MSKWDG1H = 1;     //enable high THRHLR1 threshold interrupt

//ADC_STANDBY_CH2 = Internal Channel 90;
SAR_ADC_10BIT_STDBY.ICWSELR11.B.WSEL_CH90 = 2;     //select THRHLR2 treshold for CH90
SAR_ADC_10BIT_STDBY.ICWENR2.B.WEN_CH90 = 1;     //enable watchdog for CH90
SAR_ADC_10BIT_STDBY.WTIMR.B.MSKWDG2H = 1;     //enable high THRHLR2 threshold interrupt

//ADC_STANDBY_CH3 = Internal Channel 91;
SAR_ADC_10BIT_STDBY.ICWSELR11.B.WSEL_CH91 = 3;     //select THRHLR3 treshold for CH91
SAR_ADC_10BIT_STDBY.ICWENR2.B.WEN_CH91 = 1;     //enable watchdog for CH89
SAR_ADC_10BIT_STDBY.WTIMR.B.MSKWDG3H = 1;     //enable high THRHLR3 threshold interrupt

//ADC_STANDBY_CH4 = Internal Channel 92;
SAR_ADC_10BIT_STDBY.ICWSELR11.B.WSEL_CH92 = 0;    //select THRHLR0 treshold for CH92
SAR_ADC_10BIT_STDBY.ICWENR2.B.WEN_CH92 = 1;    //enable watchdog for CH89
SAR_ADC_10BIT_STDBY.WTIMR.B.MSKWDG0H = 1;    //enable high THRHLR0 threshold interrupt

}
```

Figure 8. Standby/SSWU example timing representation (part 1)



7.13.5 SSWU contact monitoring and external watchdog refresh example

This use case example extends the previous contact monitoring use case with handling External Watchdog connected via SPI. SPI communication task has to be performed via RAM based special application code. It means that master core has to be woken up into RAM and execute watchdog refresh task. For periodical wakeup, timer can be used as RTC wakeup source (while API periodical wake-up source is dedicated for SSWU). It is assumed that MCU is preconfigured to be woken up into RAM. It is assumed that any wake-up source will be handled via special RAM based code.

Wake-up sources:

API => SSWU Contact Monitoring

RTC => External Watchdog refresh via SPI

WKUP => Any “Fast External” wake-up line (from SBC, Alarm …)

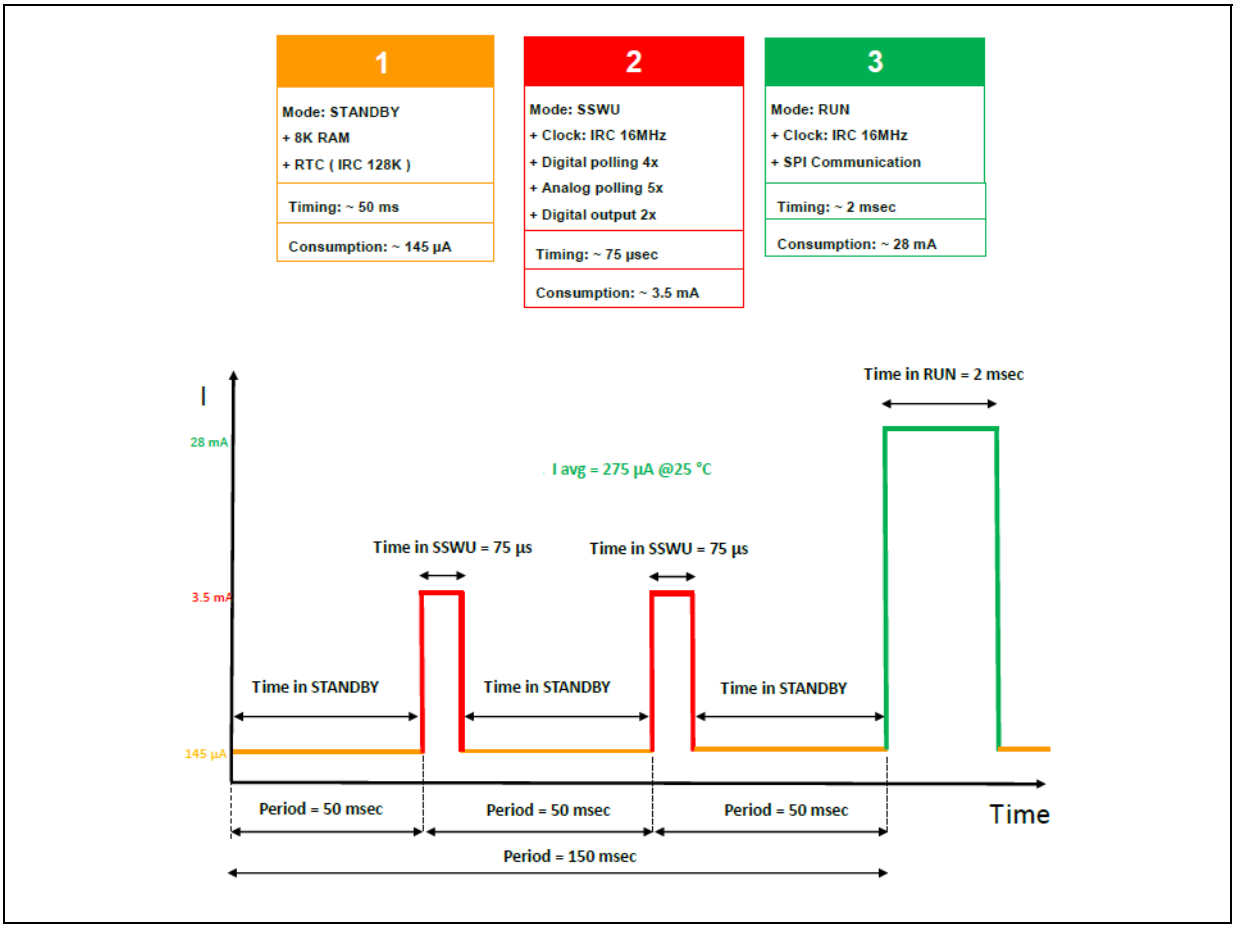
Sleep sequence:

1. Periodical contact monitoring each 50 msec via SSWU
2. Periodical external Watchdog refresh via SPI each 150 msec (via RAM based special application)

RAM Based Special Application handles all wake-up sources and decides to switch to Main flash based application if necessary.

Note: API handles next period time automatically by HW.
RTC next period time has to be SW reconfigured for next match event.

Figure 9. Standby/SSWU example timing representation (part 2)



7.13.5.1 RTCAPI initialization code sequence:

```
/*************************************************************
* FUNCTION         : RtcApi_init
*
* DESCRIPTION   : Inits RTC API for  ~ 50 ms waken up period
*                 Inits RTC for  ~ 150 ms waken up period
*                 RTC clock is 128 KHz SIRC
*
* INPUTS        :
*       - None
*
* OUTPUTS
*       - None
*************************************************************/
void RtcApi_init() {

    RTC_API.RTCC.B.CNTEN = 0;      // RTCAPI counter disabled

    RTC_API.RTCC.B.APIEN = 0;     // API disabled
    RTC_API.RTCC.B.CLKSEL = 1;    // Clk = Slow IRC ( 128 KHz nominal )
    RTC_API.RTCC.B.DIV512EN = 0;  // Divider 512 disabled
    RTC_API.RTCC.B.DIV32EN = 0;   // Divider 32 disabled
    RTC_API.RTCC.B.TRIG_EN = 1;   // SSWU self clearing wakeup API event

    RTC_API.APIVAL.B.APIVAL = API_PERIOD_VALUE;     // 1/128kHz * 6400  = 50 msec
    RTC_API.RTCVAL.B.RTCVAL = RTC_PERIOD_VALUE;   // 1/128kHz * 19200 = 150 msec

    RTC_API.RTCS.R = 0x20002400; // Clear all RTC flags

    RTC_API.RTCC.B.APIEN = 1;     // API enabled
}
```

7.13.5.2 WKPU initialization code sequence:

```
/*************************************************************
* FUNCTION      : WKPU_init
*
* DESCRIPTION   : Inits Wakeup sources
*
* INPUTS        :
*       - None
*
* OUTPUTS
*       - None
*************************************************************/
void WKPU_init(void)
{
     // Configure WKPU0 and WKPU1 ( RTC + API )
     WKPU.WRER.R     |= 3;               // Enable WKPU0 and WKUP1 for wakeup event
     WKPU.WIREER.R  |= 3;              // Rising Edge Event Enabled
```

```
// Configure WKPU6 ( Fast wakeup )
 WKPU.WRER.R    |= 1<<6;        // Enable WKPU6 for wakeup event
 WKPU.WIREER.R |= 1<<6;         // Enable raising Edge of WKUP6

 WKPU.WISR.R    = 0xFFFFFFFF;   // clear all WKUP flags
}
```

7.13.5.3 MC_ME initialization code sequence:

```
// Mode Entry and clock initialization ( code stub )

MC_ME.ME.R = 0x00002018;                  // RUN0, DRUN and STANDBY modes are enabled
RC1024K_DIG.CTL.B.LPRCON_STDBY = 1;       // Enable SIRC in STANDBY

PMCDIG.MISC_CTRL_REG.B.LPRCREG_ENB = 0;   // Enable SIRC in running modes
PMCDIG.MISC_CTRL_REG.B.RCOSC1M_ENB = 0;   // Enable SIRC in running modes

MC_ME.DRUN_MC.R          = 0x00030010;     // FLASH in normal mode, fast IRC on
MC_ME.STANDBY0_MC.R = 0x0081000F;          // FLASH in power down, fast IRC off

MC_ME.RUN_PC[1].R   = 0x00000018;          // Enable only in RUN0 and DRUN modes
MC_ME.LP_PC[1].R      = 0x00002000;        // Enable in STANDBY mode

MC_ME.PCTL[15].R      = 0x1;               // PCTL_SIUL => DBG_F = 0, LP_CFG = 1,     RUN_CFG = 1
MC_ME.PCTL[114].R    = 0x1;                // SAR_ADC10BIT_STDBY
MC_ME.PCTL[99].R      = 0x1;               // DSPI_0
MC_ME.PCTL[242].R    = 0x1;                // STDBY_CTU_0

MC_CGM.AC0_SC.R  = 0x0;                    // Select source for peripheral clock => FIRC
MC_CGM.AC0_DC0.R = 0x80000000;             // Enable divider for Peripherals
MC_CGM.AC0_DC2.R = 0x80000000;             // enable SAR_ADC_CLK

MC_CGM.AC12_SC.R  = 0x0;                   // Select source for peripheral clock => FIRC
MC_CGM.AC12_DC1.R = 0x80000000;            // Enable divider for DSPI_CLK1

MC_ME.CADDR0.R =  0x400A8000;              // Wakeup in STANDBY RAM at address 0x400A8000
```

1. MC_ME Init
2. RTCAPI Init
3. WKPU Init
4. SSWU Init
5. Wake-up address Init (MC_ME.CADDR0)
6. Copy Special RAM based application in STANDBY RAM on CADDR0 (0x400A8000)
7. Clear RTCAPI and WKPU flags
8. RTCAPI Start
9. Move to STANDBY mode (starts sleep mode sequence)

8 Typical autonomous peripherals operating while in low power mode

Low power modes architecture and implementation allow to operate the selected peripherals while in low power mode. Below you can find a list of particular examples:

HALT

- ADC – Single Shot or Scan mode
- eMIOS – PWM generation (+ CTU)
- CMU – Frequency Metering (against XOSC)
- RTC / API, PIT
- LINFLex, FlexCAN, DSPI (clocked with XOSC, PLL)

STOP

- ADC – Single Shot or Scan mode
- eMIOS – PWM generation (+ CTU)
- CMU – Frequency Metering (against XOSC)
- RTC / API, PIT
- LINFlex (with FIRC 16MHz matching 0.5%, LIN baud rate trimming according to FIRC)
- DSPI

STANDBY

- RTC / API (with SIRC 128KHz, FIRC 16MHz or SXOSC 32 kHz)
- WKPU
- SSWU
- ADC_STDBY Analog Watchdog (via SSWU)

9 Low power modes clock configurations

Table 7 shows possible clock configurations in low power modes:

Table 7. Low power modes Clock Configurations

LP MODE CLOCK	HALT	STOP	STANDBY
FIRC	DEFAULT	DEFAULT	-
XOSC	√	√	-
PLL 0 (primary)	√	-	-
PLL 1 (secondary)	√	-	-
NO CLK	-	-	DEFAULT

During STANDBY mode internal wake-up RTC / API timer could be clocked as follows:

Table 8. RTC / API STANDBY Low Power Clock

Device	SIRC 128 kHz	SXOSC 32 kHz
SPC582Bx	√	—
SPC584Bx	√	—
SPC58xCx	√	√
SPC58xGx	√	√
SPC58xHx	√	√

10 Low power modes module configurations

Table 9 shows possible module configurations in low power modes:

Table 9. Low power modes Module Configurations

MODE	PDO	MVR	FLASH	PLL 0	PLL 1	XOSC	FIRC
HALT	—	√	√	√	√	√	√
STOP	√	√	√	—	—	√	√
STANDBY	—	—	—	—	—	—	√

Table 10. STANDBY RAM Size Configurations

STANDBY MODE	8 KB RAM	32 K RAM	64 K RAM	120 K RAM	128 K RAM	256 K RAM
SPC582Bx	DEFAULT	MC_PCU.PCONF2.B.STBY0 = 1	MC_PCU.PCONF2.B.STBY0 = 1 MC_PCU.PCONF3.B.STBY0 = 1		—	—
SPC584Bx	DEFAULT	MC_PCU.PCONF2.B.STBY0 = 1	—		MC_PCU.PCONF2.B.STBY0 = 1 MC_PCU.PCONF3.B.STBY0 = 1	—
SPC58xCx	DEFAULT	MC_PCU.PCONF2.B.STBY0 = 1	—		—	MC_PCU.PCONF2.B.STBY0 = 1 MC_PCU.PCONF3.B.STBY0 = 1
SPC58xGx	DEFAULT	X	—		MC_PCU.PCONF2.B.STBY0 = 1	MC_PCU.PCONF2.B.STBY0 = 1 MC_PCU.PCONF3.B.STBY0 = 1
SPC58xHx	DEFAULT	—	—	MC_PCU.PCONF2. B.STBY0 = 1	MC_PCU.PCONF3. B.STBY0 = 256 K=	—

Note: In case of boot from STANDBY RAM after STANDBY wake-up: STANDBY RAM Boot Address has to be configured: ME_CADDR0 = 0x400A8000 (STANDBY RAM start address).

11 Low power modes consumptions

Table 11 is giving low power mode consumptions for rough estimation and comparison. Real values depend on particular derivative and configuration.

Table 11. SPC58xGx - Typical low power modes Consumptions at 25 °C

HALT	STOP	STANDBY 8K RAM
115 mA	21 mA	145 µA

Table 12. SPC58xCx - Typical low power modes Consumptions at 25 °C

HALT	STOP	STANDBY 8K RAM
74 mA	18 mA	85 µA

Table 13. SPC584Bx - Typical low power modes Consumptions at 25 °C

HALT	STOP	STANDBY 8K RAM
74 mA	18 mA	85 µA

Table 14. SPC582Bx - Typical low power modes Consumptions at 25 °C

HALT	STOP	STANDBY 8K RAM
27 mA	6.5 mA	40 µA

Table 15. SPC58xH - Typical low power modes Consumptions at 25 °C

HALT	STOP	STANDBY 8K RAM
106 mA	14.5 mA	130 mA

Note: *HALT mode:*
Flash in Low Power. Sysclk at 160 MHz, PLL0_PHI at 160 MHz, XTAL at 40 MHz, FIRC 16 MHz ON, RCOSC1M off. MCAN: instances: 0, 1, 2, 3, 4, 5, 6, 7 ON (configured but no reception or transmission), Ethernet ON (configured but no reception or transmission), ADC ON (continuously converting). All other IPs clock-gated.

STOP mode:
SYSCLK = RC16 MHz, RC16 MHz ON, RC1 MHz ON, PLL OFF. All possible peripherals off and clock gated. Flash in power down mode.

STANDBY mode:
device configured for minimum consumption, RC16 MHz off, RC1 MHz on, OSC32K off, SSWU off.

Table 16. Typical SSWU consumptions at 25 °C

Device	SSWU	SSWU with ADC
SPC58xGx	1 mA	3.5 mA
SPC58xCx	1 mA	3.5 mA
SPC584Bx	1 mA	3.5 mA
SPC582Bx	N/A	N/A
SPC58xHx	1 mA	3.5 mA

Appendix A Document management

Table 17. Reference documents

Devices	Document name	Document type
SPC58 2B line	RM0403	Reference manual
SPC58 4B line	RM0449	Reference manual
SPC58 C line	RM0407	Reference manual
SPC58 E and G line	RM0391	Reference manual
SPC58 H line	RM0452	Reference manual
SPC58 2B line	DS11597	Data sheet
SPC58 4B line	DS11701	Data sheet
SPC58 C line	DS11620	Data sheet
SPC58 E and G line	DS11758	Data sheet
SPC58 H line	DS12304	Data sheet

Revision history

Table 18. Document revision history

Date	Revision	Changes
22-May-2018	1	Initial release.
12-Jun-2018	2	Replaced all occurrences of “Chorus” with the specific root part number.
28-Jan-2022	3	Add root part number SPC58xHx. Updated *Table 1*; *Table 4*,*Table 6* , *Table 8*, *Table 10*. *Section 7.9.1*, *Section 7.10*, *Section 7.10.2*, *Section 7.10.3*. *Figure 5*. Add *Table 15*.