NXP Semiconductors
Application Note

Document Number: AN4240
Rev. 1, 10/2017

Creating Low Power Applications Using the Mode Entry Module

Configuring Qorriva Microcontrollers to Minimize Power Consumption

by: Chris Platt, Armin Winter, Carl Culshaw, and Steve Mihalik
Automotive and Industrial Solutions Group

Contents

1 Introduction

The MPC56xx family is a series of 32-bit microcontrollers based on the Power Architecture® BookE and designed specifically for embedded automotive applications.

This family contains a variety of low-power features including:

- Dynamic power management of core and peripherals
- Software-controlled clock gating of peripherals
- Multiple power domains to minimize leakage in low-power modes

This document discusses the low-power features in detail and how to use them effectively in an application to minimize the overall power used by the device. Some information out of the MPC5607BRM : MPC5607B Microcontroller–Reference Manual and AN2865: Qorivva Simple Cookbook, available on http://www.freescale.com is repeated for completeness and ease of navigation. The MPC560xB microcontroller is used in examples but in general, the features apply to other devices with the Mode Entry module.

NOTE

Some MCUs implementing the Mode Entry modules do not have all low-power modes incorporated.

2 The power problem

There are two compnents that make up the total current consumed by a microcontroller, namely dynamic and static.

2.1 Dynamic current

This is the main current prevalent during normal operating conditions. The mechanism responsible for this current is basically due to charging and discharging the gates of the millions of MOS transistors that switch on and off as the device operates. The switching currents adhere to the equation:

$$I = \frac{dQ}{dt} = C\frac{dv}{dt}$$

Thus, as technology advances and transistors become smaller, the gates become smaller resulting in lower gate capacitance. The lower capacitance means lower switching currents and hence an overall reduced dynamic run current.

The other side of the equation implies that increasing switching speeds (dv/dt) lead to an increased dynamic current.

The reduced current due to lower capacitance wins out over the increased current due to faster switching leading to a lower overall dynamic power. Unfortunately, as applications become more complex and demanding, the performance requirements and hence frequencies are increasing, leading to an overall increase in dynamic power.

2.2 Static current

This current is present when the transistors of the device are not switching and are hence in a static state. The main mechanism causing this current is leakage due to the finite resistance that exists between power and ground if power is applied to a CMOS circuit.

This leakage current is highly dependent on the threshold voltage of the transistor. As advances in technology lead to ever smaller device sizes, supply voltage levels are scaled lower. To improve circuit speed, the threshold voltages are also decreased, resulting in an exponential increase in sub-threshold leakage current. In summary, as technology shrinks the device size, even though dynamic power trends lower, the required increase in performance means that any savings made by technology are neutralized by the increased frequencies. So, overall power is trending upward.

Couple this fact with a large increase in static leakage inherent due to smaller device sizes. So, the overall dynamic power increases with the static current becoming a larger component of the overall current.

3 Low-Power feature summary

Because the low-power requirements are showing an upward trend, the MPC56xxB/C/D and several other Qorivva family members have a number of features that are specifically designed to minimize power consumption. This is achieved by influencing both the dynamic as well as the static constituent parts of power consumption.

Table 1. Low-power mode feature assignment

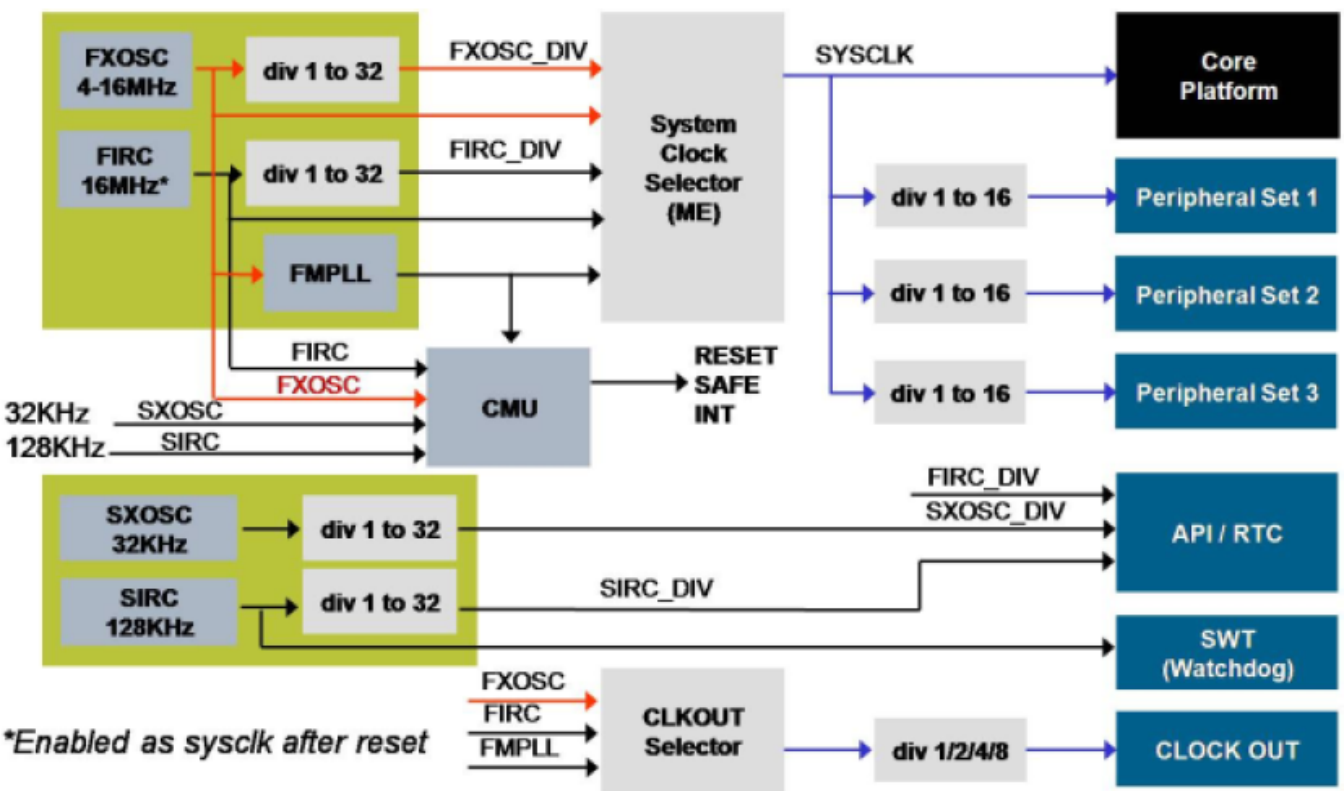
Low-power feature	Static power consumption	Dynamic power consumption
Clock management		X
The ability to stop clocks on a per module basis		X

Table continues on the next page...

Table 1. Low-power mode feature assignment (continued)

Optimized clock tree design		X
Ability to divide down the system clock to peripherals		X
Ability to gate off or divide down the clock to the core		X
Onboard clocking		X
Part can self-clock without a FMPLL		X
16 MHz internal oscillator		X
128 kHz internal oscillator		X
Power gating	X	
Removing power to large areas of silicon	X	
Active well biasing (in Stop mode)	X	
Dual core (available on some derivatives)		X
Able to run at lower speed to achieve the same throughput		X

4 Clock description



Power consumption is directly linked to clock signal switching, that is, the power consumption increases if more gates are switching. Within the MPC56xxB family, several methods are employed to avoid “wasting” power in clock edges:

- Clock freeze
  - CPU activity can be temporarily halted, for example, while waiting for an analog-to-digital conversion (ADC) to complete. This is implemented using WAIT instruction, or STOP and HALT modes.
- Peripheral bus division
  - Using dividers in the Clock Generation module, can reduce clock rates, for example, in ADC and communication ports. All the modules don’t need to run at full system speed; the clocks can be slowed down in each module to save power.
- Clock Gating
  - Applied wherever possible and at the entry to each sub-module (peripheral). Switch off the clocks to any unused module to save power. This is implemented using peripheral configurations in the Mode Entry module.

The MPC56xxB clock structure allows system clock selection and dividing clocks to most of the peripherals.

Table 2. Clock source behavior summary

Clock source	Start-up time	Power consumption	Check errors
FIRC: 16 MHz internal RC oscillator .	Low (< 1 µs)	Medium (< 50 µA)	Low (> 10%)
SIRC: 128 kHz internal RC oscillator	Low (< 1 µs)	Low (<1 µA)	Low (> 10%)
SXOSC: 32 kHz external crystal oscillator .	High (ms)	Low (<10 µA)	High (< 1%)
FXOSC: 4–16 MHz external crystal oscillator .	High (ms)	High (>100 µA)	High (< 1%)
FMPLL	High (ms)	High (>1 mA)	High (< 1%)

5 Power control unit

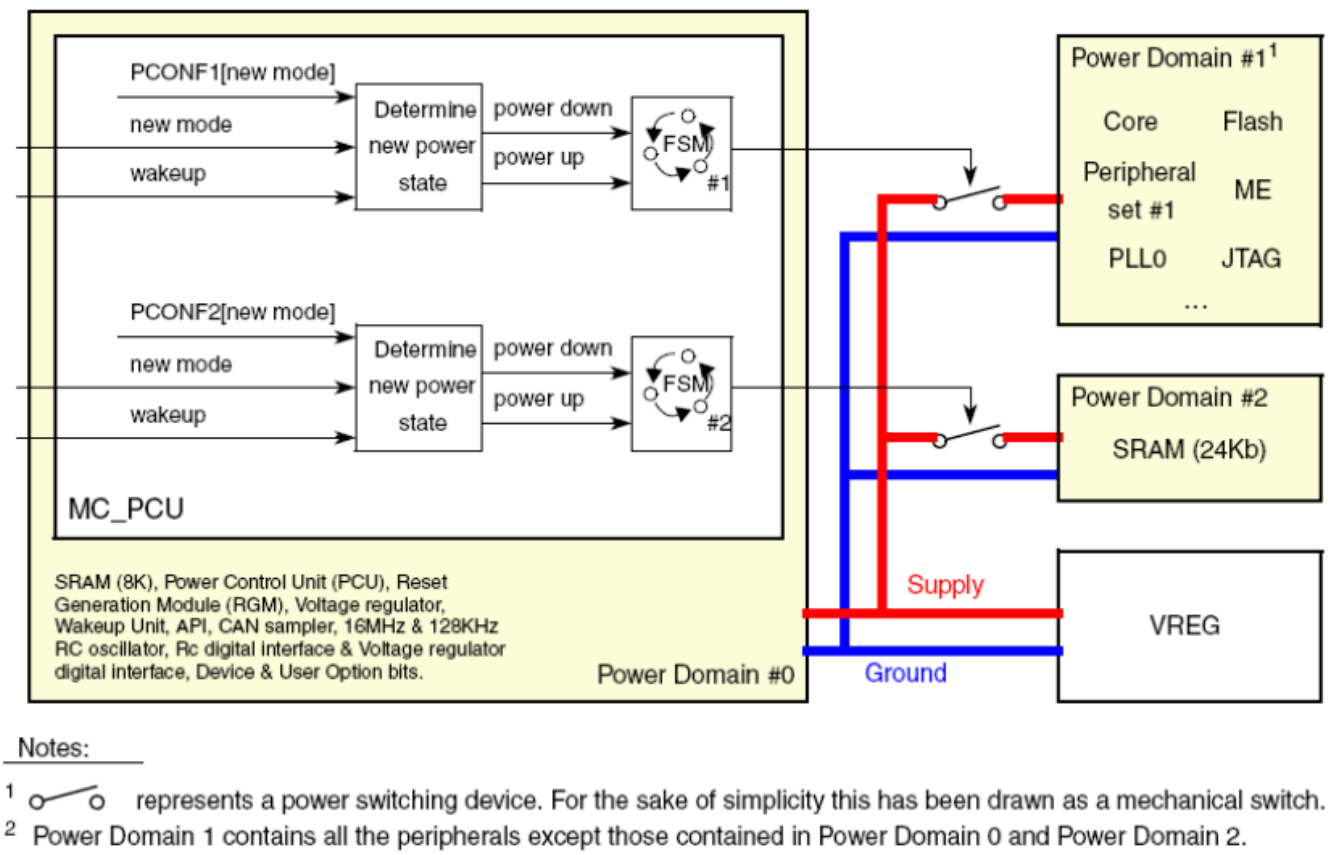
The purpose of the power control unit (PCU) is to reduce overall device power consumption. The MCU peripherals are allocated to various power domains. The PCU allows users to remove or apply power to a power domain (PD) depending on the operating mode.

Hardware is used to control switching power on and off to primary power domains (PD0 and PD1) based on the mode being entered to conserve current. When a domain is powered off, then all initialization and other data is lost for registers and memory in that domain. Unlike power domains PD0 and PD1, power for PD2 is configurable by software writing to the PCU_PCONF2 register.

Going to STANDBY mode cuts power to the core, most of the peripherals, mode entry configurations, clocks, and so on. Therefore, registers in PD0 and PD1 need to be reinitialized when exiting the STANDBY mode, but not for other modes such as STOP mode.

5.1 Power domain control

- Power domains are controlled on a device mode basis.
- For each mode, software can configure whether certain power domains are connected to the supply voltage during the power-up state, or disconnected during the power-down state.
- Maximum power saving is reached by entering the STANDBY mode.
- On each mode change request, the MC_PCU evaluates the power domain settings in the power domain configuration registers and initiates a power-down or a power-up sequence for each individual power domain.
- The power-up/down sequences are handled by finite state machines to ensure a smooth and safe transition from one power state to the other.
- Exit from STANDBY mode can be made only via a system wake-up event as all power domains other than PD0 are in the power-down state.



Notes:

1 ⟋ represents a power switching device. For the sake of simplicity this has been drawn as a mechanical switch.

2 Power Domain 1 contains all the peripherals except those contained in Power Domain 0 and Power Domain 2.

Figure 1. MPC5604/3/2B/C PCU block diagram

Table 3. MPC5604/3/2B/C PCU setup

Power Domain	Contains	State
PD0	Minimal circuitry for low power, including first 8 KB SRAM, WKPU Unit, CAN Sampler, RTC, API, etc.	Always ON

Table continues on the next page...

Table 3. MPC5604/3/2B/C PCU setup (continued)

Power Domain	Contains	State
PD1	The rest of the device except SRAM	STANDBY: OFF
		Other Modes: ON
PD2	SRAM other than the first 8 KB SRAM	Configurable for each mode

All memories which are not powered down during STANDBY mode automatically enter a power saving state. No software configuration is required to enable this power-saving state.

NOTE

PCU setup of other derivatives might contain an additional power domain with a slightly different configuration. See the reference manual of the corresponding device of the MPC56xx family, available on http://www.freescale.com.

6 Mode Entry (ME) module

6.1 Overview

The Mode Entry (ME) module controls the device mode and mode transition sequences in all functional states. It also contains configuration, control and status registers accessible for the application. By supporting a variety of modes, it accommodates different application needs. Each mode is configurable and can be setup to achieve an optimum between power management and system performance requirements.

Modes can be preconfigured during device initialization and mode changes are triggered by a simple software (SW) sequence. Therefore, it offers a centralized and easy way to switch between modes depending on application needs.

An additional low-power mode configuration example is given within application note AN2865: Qorivva Simple Cookbook, which is available for download from http://www.freescale.com.

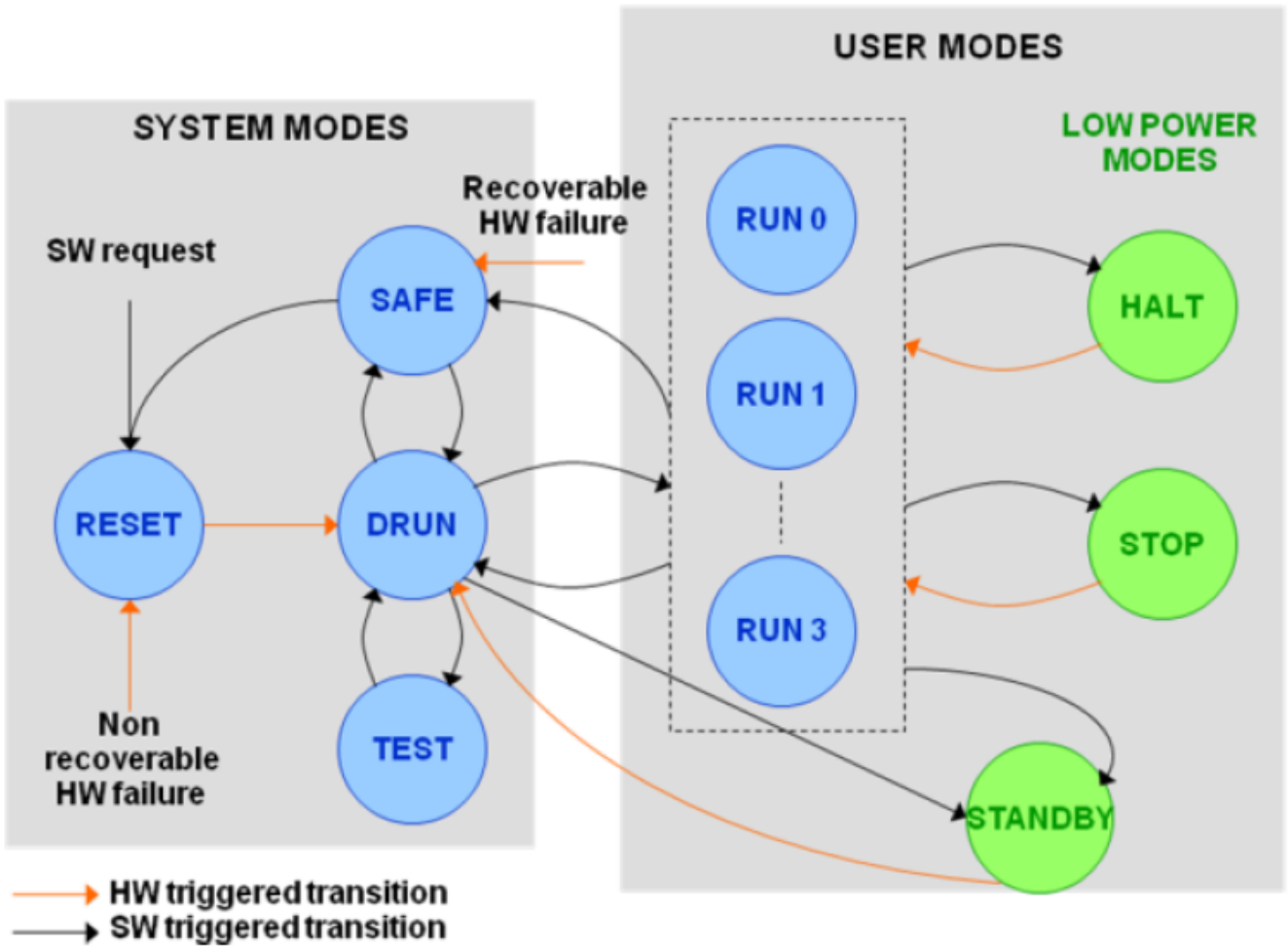


Figure 2. Mode diagram

6.2 Low-power modes

6.2.1 HALT mode

HALT mode is a reduced activity, low-power mode intended for moderate periods of lower processing activity, such as during a slow serial communication like LIN frame transmission or reception. In this mode, the core system clocks are stopped but user-selected peripheral (LINFlex) tasks can continue to run. It can be configured to provide more efficient power management features such as switch-off FMPLL, flash memory, and main regulator, at the cost of longer wake-up latency. The system returns to RUN mode as soon as an event or interrupt is pending.

The key features of HALT mode are as follows:

- The core is stopped but system clock can remain the same as in RUN mode.
- Flash can be put in low-power mode.
- Peripherals can continue to run at normal or reduced frequency.
- This mode is useful to reduce device consumption during a slow serial communication such as LIN frame transmission or reception.

- Analog device components such as phase-locked loop (PLL), analog-to-digital converter (ADC), and potentially the main voltage regulator, can be stopped.
- PLL can be configured to be on or off (default is off).

6.2.2 WAIT instruction

The MPC56xx e200 CPUs support a WAIT instruction. Executing this WAIT instruction has a similar effect as entering HALT mode. By executing the WAIT instruction, only the CPU clock gets stopped, whereas in HALT mode, the core platform clock gets stopped which causes additional functionality like System Timer Module (STM), Crossbar Switch (XBAR), and other modules, to be halted. As a consequence, current consumption after executing the WAIT instruction is higher compared to HALT mode.

This is the only low-power mode not configured and entered by the ME module. With only four system clock cycles, a very short wake-up latency is supported.

6.2.3 STOP mode

STOP mode maintains power to the entire device allowing the retention of all on-chip registers and memory, and providing a faster recovery low-power mode than STANDBY mode which offers the lowest power consumption. There is no need to reconfigure the device before executing code. The clocks to the core and peripherals are halted and can be optionally stopped at the expense of a slower startup time. FMPLL is always disabled during STOP mode. During STOP mode exit, the FIRC is used to clock the part until the target clock is available.

STOP is entered from RUN mode only. Wake-up from STOP mode is triggered by an external event or by the internal periodic wake-up, if enabled.

The key features of STOP mode are as follows:

- The core provides additional low-power features beyond HALT.
- Active clock gating is used to segment the device.
- External oscillator is stopped, but can be allowed to run to support fast startup at the expense of added power.
- The entire device is powered and RAM is retained.
- RTC and API can continue to run.
- PLL is always off.
- It has the option to support fast and slow IRC.
- System clock can be disabled.
- PLL is always disabled.
- System clock is only FIRC upon immediate STOP mode exit until the target clock is ready.

6.2.4 STANDBY mode

STANDBY mode halts the clock to the entire device and turns off the power to the majority of the chip to offer the lowest possible power consumption.

The device can be woken up from STANDBY mode by any of the external wake-up pins, a reset, or from a periodic wake-up using a low-power oscillator. If required, it is possible to enable the internal 16 MHz or 128 kHz RC oscillator or external 32 kHz oscillator (not available on all derivatives). In STANDBY mode, the contents of the cores, on-chip peripheral registers and potentially some of the volatile memory are not held.

Depending on the user configuration, either 32 KB, or 8 KB of SRAM are retained (MPC5604B). A fast wake-up using the on-chip 16 MHz internal RC oscillator allows rapid execution from RAM on exit from low-power modes. This oscillator supports low-speed code execution and clocking of peripherals through selection as the system clock, and it can be used as the FMPLL input clock source to provide fast startup without the external oscillator delay.

In low-power modes, the internal 16 MHz RC oscillator also supports the operation of ADCs. External wake-up pins are available for wake-up, and a fast startup internal voltage regulator provides a rapid exit from low-power modes.

The key features of STANDBY mode are:

- This mode consumes the lowest possible power.
- Most functions (digital and analog) of the device are not powered.
- Remains only the back-up logic. For example, RTC/API, preserves wake-up inputs, part of SRAM.
- On STANDBY exit, the processor uses the RESET vector or a SRAM Vector, if enabled.
- Wake-up from selected I/O or API/RTC
- WISR register in the WKPU can be used to verify the wake-up source.
- Data saved prior to mode entry can be recovered on exit from the mode, if needed.
- Selectable size of RAM supported: 8 KB or all RAM
- Optionally enabled low-speed IRC

6.3 Low-Power Peripheral Configuration registers

Each peripheral can be associated with a particular clock gating policy. The policy is determined by two groups of peripheral configuration registers:

- ME_RUN_PC0:7 for RUN modes
- ME_LP_PC0:7 for low-power modes

Clocks to peripherals are gated off unless enabled for that mode.

Address 0xC3FD_C0A0 - 0xC3FD_C0BC Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STANDBY	0	0	STOP	0	HALT	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 3. Low-Power Peripheral Configuration registers (ME_LP_PC0…7)

Along these registers, a selection of eight low-power mode configurations for peripherals can be defined. There is an equivalent register set (ME_RUN_PC0…7) for run modes available.

Table 4. Low-Power Peripheral Configuration registers (ME_LP_PC0…7) field descriptions

Field	Description
STANDBY	Peripheral control during STANDBY 0 Peripheral is frozen with clock gated. 1 Peripheral is active.
STOP	Peripheral control during STOP

Table continues on the next page...

Table 4. Low-Power Peripheral Configuration registers (ME_LP_PC0…7) field descriptions (continued)

Field	Description
	0 Peripheral is frozen with clock gated. 1 Peripheral is active.
HALT	Peripheral control during HALT 0 Peripheral is frozen with clock gated. 1 Peripheral is active.

6.4 Peripheral Control register

For each on-chip peripheral, a Peripheral Control (ME_PCTLx) register exists to control clock gating for the particular peripheral.

These registers select one of the 8 RUN peripheral configurations for run (RUN_CFG), and one of 8 low power peripheral configurations (LP_CFG). Additionally, there is an option to enable/disable freezing the clock during debugging (DBG_F) for each peripheral.

Figure 4. Peripheral Control registers (ME_PCTL0…143)

Table 5. Peripheral Control registers (ME_PCTL0…143) field descriptions

Field	Description
DBG_F	Peripheral Control in Debug Mode — This field controls the state of peripheral in Debug mode. 0 Peripheral state depends on RUN_CFG/LP_CFG bits and the device mode. 1 Peripheral is frozen if not already frozen in device modes. **NOTE:** This feature is useful to freeze the peripheral state while entering Debug mode. For example, this may be used to prevent a reference timer from running while making a debug access.
LP_CFG	Peripheral Configuration Select for Non-RUN modes—This field associates a configuration as defined in the ME_LP_PC0…7 registers to the peripheral. 000 Selects ME_LP_PC0 configuration 001 Selects ME_LP_PC1 configuration

Table continues on the next page...

Table 5. Peripheral Control registers (ME_PCTL0…143) field descriptions (continued)

Field	Description
	010 Selects ME_LP_PC2 configuration 011 Selects ME_LP_PC3 configuration 100 Selects ME_LP_PC4 configuration 101 Selects ME_LP_PC5 configuration 110 Selects ME_LP_PC6 configuration 111 Selects ME_LP_PC7 configuration
RUN_CFG	Peripheral Configuration Select for RUN modes—This field associates a configuration as defined in the ME_RUN_PC0…7 registers to the peripheral. 000 Selects ME_RUN_PC0 configuration 001 Selects ME_RUN_PC1 configuration 010 Selects ME_RUN_PC2 configuration 011 Selects ME_RUN_PC3 configuration 100 Selects ME_RUN_PC4 configuration 101 Selects ME_RUN_PC5 configuration 110 Selects ME_RUN_PC6 configuration 111 Selects ME_RUN_PC7 configuration

6.5 Peripheral Status register

This set of read-only registers provides the status of the peripherals.

Address 0xC3FD_C060 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	S_DMA_CH_MUX	0	S_FlexCAN5	S_FlexCAN4	S_FlexCAN3	S_FlexCAN2	S_FlexCAN1	S_FlexCAN0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	S_LINFlex9	S_LINFlex8	0	0	S_DSPI5	S_DSPI4	S_DSPI3	S_DSPI2	S_DSPI1	S_DSPI0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 5. Peripheral Status registers (ME_PS0…3)

Table 6. Field description

Field	Description
S_<periph>	Peripheral Status —This field specifies the current status of the peripherals in the system. If no peripheral is mapped on a particular position, the corresponding bit is always read as 0. 0 Peripheral is frozen. 1 Peripheral is active.

6.6 Mode Enable register

Before entering, all the modes are enabled using the Mode Enable (ME_ME) register. An interrupt is generated if attempt is made to enter a disabled mode.

This register allows to disable modes not required for a given application. RESET, SAFE, DRUN and RUN0 modes are always enabled.

Address 0xC3FD_C008 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	STANDBY	0	0	STOP	0	HALT	RUN3	RUN2	RUN1	RUN0	DRUN	SAFE	TEST	RESET
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1

Figure 6. Mode Enable (ME_ME) register

Table 7. Mode Enable (ME_ME) register field descriptions

Field	Description
STANDBY	STANDBY mode enable 0 STANDBY mode is disabled. 1 STANDBY mode is enabled.
STOP	STOP mode enable 0 STOP mode is disabled. 1 STOP mode is enabled.
HALT	HALT mode enable

Table continues on the next page...

Table 7. Mode Enable (ME_ME) register field descriptions (continued)

Field	Description
	0 HALT mode is disabled. 1 HALT mode is enabled.
RUN3	RUN3 mode enable 0 RUN3 mode is disabled. 1 RUN3 mode is enabled.
RUN2	RUN2 mode enable 0 RUN2 mode is disabled. 1 RUN2 mode is enabled.
RUN1	RUN1 mode enable 0 RUN1 mode is disabled. 1 RUN1 mode is enabled.
RUN0	RUN0 mode enable 0 RUN0 mode is disabled. 1 RUN0 mode is enabled.
DRUN	DRUN mode enable 0 DRUN mode is disabled. 1 DRUN mode is enabled.
SAFE	SAFE mode enable 0 SAFE mode is disabled. 1 SAFE mode is enabled.
TEST	TEST mode enable 0 TEST mode is disabled. 1 TEST mode is enabled.
RESET	RESET mode enable 0 RESET mode is disabled. 1 RESET mode is enabled.

6.7 Software (SW) and Hardware (HW) mode transitions

Software handled transition involves the following steps:

- A transition is requested by writing a key protected sequence in the ME_MCTL register.
- ME_MCTL needs to be written twice:
  - 1st write: TARGET_MODE + KEY
  - 2nd write: TARGET_MODE + INVERTED KEY

The S_MTRANS field within the Global Status (ME_GS) register, or ME_GS[S_MTRANS] indicates that a mode change transition is in progress. This field is cleared when the transition is complete.

```
/* Enter HALT Mode */
       ME.MCTL.R = ((vuint32_t)HALT_MODE<<28) | 0x00005AF0;      /* Mode & Key */
       ME.MCTL.R = ((vuint32_t)HALT_MODE<<28) | 0x0000A50F;      /* Mode & Key */
       while (ME.GS.B.S_MTRANS) {}    /* Wait HALT mode transition to complete */
                                      /* For series production code, a SW      */
                                      /* timeout is recommended                */
```

The I_MTC field within the Interrupt Status Register (ME_IS), or ME_IS[I_MTC] also indicates a completed mode transition between DRUN/RUNx modes. ME_IS[I_MTC] is not set in case of transition to low-power modes excluding execution of WAIT instruction.

- Mode Entry configures the modules according to the ME_xxx_MC register of the target mode
- Mode transition does not complete until modules are ready, including
  - OSC, if turned on, completes OSC specified timeout.
  - PLL, if turned on, is locked.
- The status bit/interrupt signals the completion of transition.

NOTE

Modification of a ME_xxx_MC register (even the current one) is taken into account on next mode “xxx” entry.

Hardware triggered transition involves:

- Exit from low-power mode
- SAFE transition caused by HW failure
- RESET transition caused by HW failure

6.8 Mode Global Status register

This register contains the global mode status.

It is recommended that software poll ME_GS[S_MTRANS] after requesting a transition to HALT, STOP, or STANDBY.

Address 0xC3FD_C000 Access: User read, Supervisor read, Test read

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	S_CURRENT_MODE				S_MTRANS	S_DC	0	0	S_PDO	0	0	S_MVR	S_DFLA		S_CFLA	
W																
Reset	0	0	0	0	1	1	0	0	0	0	0	1	1	1	1	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	S_FMPLL	S_FXOSC	S_FIRC	S_SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 7. Global Status (ME_GS) register

Table 8. Field description of ME_GS register

Field	Description
S_CURREN T_MODE	Current Device Mode Status 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT 1001 Reserved 1010 STOP 1011 Reserved 1100 Reserved 1101 STANDBY 1110 Reserved 1111 Reserved
S_MTRANS	Mode Transition Status 0 Mode transition process is not active. 1 Mode transition is ongoing.
S_DC	Device Current Consumption Status 0 Device consumption is low enough to allow powering down of main voltage regulator. 1 Device consumption requires main voltage regulator

Table continues on the next page...

Table 8. Field description of ME_GS register (continued)

Field	Description
	to remain powered regardless of mode configuration.
S_PDO	Output Power-Down Status—This field specifies output power-down status of I/Os. This field is asserted whenever outputs of pads are forced to high-impedance state or the pads power sequence driver is switched off. 0 No automatic safe-gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high-impedance state and pads power sequence driver is disabled. The levels of inputs are unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept unchanged. In STANDBY mode, the power sequence driver and all pads except those mapped on wake-up lines are not powered and are therefore in high-impedance state. Wake-up lines configuration remains unchanged.
S_MVR	Main Voltage Regulator Status 0 Main voltage regulator is not ready. 1 Main voltage regulator is ready for use.
S_DFLA	Data Flash Availability Status 00 Data flash is not available. 01 Data flash is in power-down mode. 10 Data flash is in low-power mode. 11 Data flash is in normal mode and available for use.
S_CFLA	Code Flash Availability Status 00 Code flash is not available. 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode and available for use.
S_FMPLL	Frequency-Modulated Phase-Locked Loop Status 0 Frequency-modulated phase locked loop is not stable. 1 Frequency-modulated phase locked loop is providing a stable clock.
S_FXOSC	Fast External Crystal Oscillator (4–16 MHz) status 0 Fast external crystal oscillator (4–16 MHz) is not stable 1 Fast external crystal oscillator (4–16 MHz) is providing a stable clock
S_FIRC	Fast Internal RC Oscillator (16 MHz) Status

Table continues on the next page...

Table 8. Field description of ME_GS register (continued)

Field	Description
	0 Fast internal RC oscillator (16 MHz) is not stable. 1 Fast internal RC oscillator (16 MHz) is providing a stable clock.
S_SYSCLK	System Clock Switch Status — This field specifies the system clock currently used by the system. 0000 16 MHz internal RC oscillator 0001 div. 16 MHz internal RC oscillator 0010 4–16 MHz external crystal oscillator 0011 div. external crystal oscillator 0100 Frequency-modulated phase-locked loop 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved 1110 Reserved 1111 System clock is disabled.

6.9 Mode Control register

This register is used to trigger software-controlled mode changes. In order to change modes, the target mode must be written to the register with the KEY. The next command must be another write to the register with the same mode and the INVERTED KEY.

Address 0xC3FD_C004 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	TARGET_MODE				0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1
W	KEY															
Reset	1	0	1	0	0	1	0	1	0	0	0	0	1	1	1	1

Figure 8. Mode Control (ME_MCTL) register

Table 9. Mode Control (ME_MCTL) register field descriptions

Field	Description
TARGET_MODE	Target Device Mode This field provides the target device mode to be entered by software programming. The mechanism to enter into any mode by software requires the write operation twice: first time with key, and second time with inverted key. This field is automatically updated by hardware while entering SAFE on hardware request. Also, while exiting from the HALT and STOP modes on hardware exit events, this field is updated with the appropriate RUN0…3 mode value. 0000 RESET 0001 TEST 0010 SAFE 0011 DRUN 0100 RUN0 0101 RUN1 0110 RUN2 0111 RUN3 1000 HALT 1001 Reserved 1010 STOP 1011 Reserved 1100 Reserved 1101 STANDBY 1110 Reserved 1111 Reserved
KEY	Control Key This field enables write access to this register. Any write access to the register with a value different from the keys is ignored. Read access will always return the inverted key. KEY: 0101101011110000 (0x5AF0) INVERTED KEY: 1010010100001111 (0xA50F)

6.10 Mode Entry module summary

Mode transition allows:

- Enabling/disabling system clock sources
- Selecting appropriate system clock
- Gating clocks to peripherals
- Dividing peripheral clocks on a set basis, as per the requirement

Resource	Mode							
	RESET	TEST	SAFE	DRUN	RUN[0:3]	HALT	STOP	STANDBY
FIRC	on	✓ on	on	on	on	✓ on	✓ on	✓ on
FXOSC	off	✓ off	off	✓ off	✓ off	✓ off	✓ off	off
FMPLL	off	✓ off	off	✓ off	✓ off	✓ off	off	off
CFLASH	normal	✓ normal	normal	✓ normal	✓ normal	✓ low-power	✓ power-down	power-down
DFLASH	normal	✓ normal	normal	✓ normal	✓ normal	✓ low-power	✓ power-down	power-down
MVREG	on	on	on	on	on	✓ on	✓ on	off
PDO	off	✓ off	✓ on	off	off	off	✓ off	on

✓ Configurable

Not configurable

Figure 9. ME resource control overview

System Clock Source	Mode							
	RESET	TEST	SAFE	DRUN	RUN[0:3]	HALT	STOP	STANDBY
FIRC (16MHz) internal	✓ default	✓ default	✓ default	✓ default	✓ default	✓ default	✓ default	
FIRC (16MHz) internal divided		✓		✓	✓	✓	✓	
FXOSC (4...16MHz) external		✓		✓	✓	✓		
FXOSC (4...16MHz) ext. divided		✓		✓	✓	✓	✓	
FMPLL		✓		✓	✓	✓		
System Clock disabled		✓ 1					✓	✓ default

✓ Configurable

Not configurable

1 Disabling the system clock in TEST mode will require a RESET to exit TEST mode

Figure 10. ME system clock source overview

NOTE

If the user steps through the code that writes to a peripheral register but the debugger shows the register unchanged, the clock for that peripheral in that particular RUN mode may not have been enabled.

7 Pins during low-power modes

7.1 STANDBY mode

In STANDBY Mode, pins other than wake-up inputs with enabled pull resistors are always in high-impedance state. External voltages in the range of 0–5 V do not generate additional current consumption. It is important to remember that all pins must have a pullup/pulldown either internally with WPU or externally.

Unconnected GPIOs do have a weak internal pull active, preventing the pin from floating in STANDBY.

NOTE

The statements given above regarding STANDBY do not apply to wake-up pins. For wake-up pins, the internal pull selection needs to be aligned to the external voltages to avoid additional current consumption.

In summary, all wake-up pads, including those not bonded out to pins, must be correctly configured to achieve low-power usage in STANDBY mode. GPIOs without wake-up capability do not need special configuration for STANDBY mode.

Pins having both the wake-up and analog functionality such as PB[10], PD[0:1] on MPC5607/6/5B/C and MPC5602/1D, or PB[10] on MPC5604/3/2B/C, must be driven either high-level or low-level (possibly using the internal pullup) during STANDBY.

In case these pins are connected to external component providing analog signal, it is important to check this external analog signal is either lower than 0.2*VDD_HV or higher than 0.8*VDD_HV not to incur extra consumption.

The TDO pad has been moved into the STANDBY domain (PD0) in order to allow low-power debug handshaking in STANDBY mode. However, no pullup resistor is active on the TDO pad while in STANDBY mode. At this time the pad is configured as an input. When no debugger is connected, the TDO pad is floating causing additional current consumption.

To avoid the extra consumption, TDO must be connected. An external pullup resistor in the range of 47–100 kΩ must be added between the TDO pin and VDD. If the TDO pin is used as application pin and a pullup cannot be used, then a pulldown resistor with the same value must be used between TDO pin and GND instead.

On MPC5604/3/2/B/C, wake-up line functionality on PB[8], PB[9] is not available in STANDBY mode. These pads are not supplied with ultra low-power regulator, but are driven from main regulator which is switched off in STANDBY mode.

7.2 STOP mode

During STOP mode, status of pins will be kept unchanged. In this case, the internal pull selection needs to be aligned to the external voltages to avoid additional current consumption in STOP mode.

Leaving input pins floating will cause the pin voltage to float to any voltage depending upon leakages to ground and the supply. Noise will cause the CMOS inverter or gate to which the input is connected to go into an indefinite and high-power dissipation mode, with both N- and P-channel outputs conducting.

Based on this effect, external noise can cause the input stage to switch states at high-frequency (oscillations). Switching from one state to the other will add additional current consumption to the static leakage current. Also, an increased level of noise gets injected to the microcontroller.

8 Low-Power mode entry

Low-power mode entry is controlled by software requests only.

CAUTION

The ME module will not allow low-power mode entry, if it detects a pending wake-up event or pending interrupt request. While examining for a pending interrupt request, the ME module ignores the MSR[EE] state and peripheral's interrupt priorities in the interrupt controller (INTC). The ME module effectively looks at peripheral flags and their associated interrupt enable bit state.

The following table lists various RUN modes from which the low-power modes can be entered.

Table 10. Low-power mode entry

	WAIT	STOP	HALT	STANDBY
Software request from DRUN	Yes	No	No	Yes
Software request from RUN0...3	Yes	Yes	Yes	Yes

Note

HPREG can be turned off in STOP and HALT mode entry by monitoring S_DC bit via the MC_ME mode transition. However, this bit can only be used during this transition. Using it in any other condition will have unexpected behavior. This is only applicable for MPC5604B and MPC5607B devices.

8.1 HALT mode entry

HALT mode can be requested by SW from RUN0…3 modes.

For example:

```
void HALT_Mode_Entry(void)
{
    /* Enter HALT Mode */
    ME.MCTL.R = 0x80005AF0;       /* Mode & Key */
    ME.MCTL.R = 0x8000A50F;       /* Mode & Key */
    while (ME.GS.B.S_MTRANS) {} /* Wait HALT mode transition to complete */
}
```

On MPC560xB/C/D derivatives, main VREG is not disabled during STOP or HALT mode:

- If RUN[0..3] mode selects FXOSC to be running.
- The target mode selects FXOSC as system clock.

If STOP or HALT is configured with:

- ME_[mode]_MC.MVRON = 0
- ME_[mode]_MC.FIRCON = 0, and
- ME_[mode]_MC.SYSCLK = 0010/0011, the main VREG will nevertheless remain enabled during the STOP mode if the previous RUN[0..3] mode is configured with ME_RUN[0..3]_MC.FXOSCON = 1. This will result in higher power consumption than expected.

Before entering STOP or HALT mode with the following configuration:

- ME_[mode]_MC.MVRON = 0
- ME_[mode]_MC.FIRCON = 0
- ME_[mode]_MC.SYSCLK = 0010/0011, ensure the RUN[0..3] mode switches off FXOSC and ME_RUN[0..3]_MC.FXOSCON = 0 before attempting to enter the low-power mode.

8.2 STOP mode entry

STOP mode can be requested by SW from RUN0…3 modes.

For example:

```
void STOP_Mode_Entry(void)
{
    /* Enter STOP Mode */
    ME.MCTL.R = 0xA0005AF0;       /* Mode & Key */
    ME.MCTL.R = 0xA000A50F;       /* Mode & Key */
    while (ME.GS.B.S_MTRANS) {} /* Wait STOP mode transition to complete */
```

8.2.1 Mode Entry STOP Mode Configuration register

NOTE

There is only one STOP mode, but most of the SRAM can be configured to either have its power maintained or not. STOP0 is a term often used to indicate STOP mode with most of the SRAM unpowered.

The STOP Mode Configuration (ME_STOP_MC) register configures system behavior during STOP mode.

Byte and half-word write accesses are not allowed to this register.

Address 0xC3FD_C048 Access: User read, Supervisor read/write, Test read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	PDO	0	0	MVRON	DFLAON		CFLAON	
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	0	0	0	0	0	0	0	0	FMPLLON	FXOSCON	FIRCON	SYSCLK			
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Figure 11. STOP Mode Configuration (ME_STOP_MC) register

Table 11. Field description of ME_STOP_MC

Field	Description
PDO	I/O Output Power-Down Control—This field controls the output power-down of I/Os. 0 No automatic safe-gating of I/Os used and pads power sequence driver is enabled. 1 In SAFE/TEST modes, outputs of pads are forced to high-impedance state and pads power sequence driver is disabled. The levels of inputs are unchanged. In STOP mode, only pad power sequence driver is disabled but the state of the output is kept unchanged. In STANDBY mode, power sequence driver and all pads except those mapped on wake-up lines are not powered and therefore are in high-impedance state. Wake-up line configuration remains unchanged.

Table continues on the next page...

Table 11. Field description of ME_STOP_MC (continued)

Field	Description
MVRON	Main Voltage Regulator Control—This field specifies whether main voltage regulator is switched off or not while entering this mode. 0 Main voltage regulator is switched off. 1 Main voltage regulator is switched on.
DFLAON	Data Flash Power-Down Control—This field specifies the operating mode of the data flash after entering this mode. 00 Reserved 01 Data flash is in power-down mode. 10 Data flash is in low-power mode. 11 Data flash is in normal mode.
CFLAON	Code Flash Power-Down Control—This field specifies the operating mode of the program flash after entering this mode. 00 Reserved 01 Code flash is in power-down mode. 10 Code flash is in low-power mode. 11 Code flash is in normal mode.
FMPLLON	Frequency-Modulated Phase Locked Loop Control 0 Frequency-modulated phase-locked loop is switched off. 1 Frequency-modulated phase-locked loop is switched on.
FXOSCON	Fast External Crystal Oscillator (4–16 MHz) control 0 Fast external crystal oscillator (4–16 MHz) is switched off. 1 Fast external crystal oscillator (4–16 MHz) is switched on.
FIRCON	Fast Internal RC Oscillator (16 MHz) Control 0 Fast internal RC oscillator (16 MHz) is switched off. 1 Fast internal RC oscillator (16 MHz) is switched on.
SYSCLK	System Clock Switch Control—This field specifies the system clock to be used by the system. 0000 16 MHz internal RC oscillator 0001 div. 16 MHz internal RC oscillator 0010 4–16 MHz external crystal oscillator 0011 div. external crystal oscillator 0100 Frequency-modulated phase-locked loop 0101 Reserved 0110 Reserved 0111 Reserved 1000 Reserved 1001 Reserved 1010 Reserved 1011 Reserved 1100 Reserved 1101 Reserved

Table 11. Field description of ME_STOP_MC

Field	Description
	1110 Reserved 1111 System clock is disabled.

Lowest possible STOP power consumption is achieved by following sequence:

```
void STOP_Mode_Entry(void)
{
ME_STOP_MC.B.PDO=1;      /* Pad power sequence driver disabled, */
                          /* but state of the output is kept */
ME_STOP_MC.B.MVRON=0;    /* Main Voltage regulator is off */ ME_STOP_MC.B.DFLAON=1;   /*
Data flash is in power-down mode */
ME_STOP_MC.B.CFLAON=1;   /* Code flash is in power-down mode */
ME_STOP_MC.B.FMPLLON=0;  /* FMPLL off (default status during STOP) */
ME_STOP_MC.B.FIRCON=0;   /* Fast internal RC oscillator off */
ME_STOP_MC.B.FXOSCON=0   /* Fast external crystal oscillator off */
ME_STOP_MC.B.SYSCLK=0xF; /* System clock is disabled */
ADC_0.MCR.B.PWDN = 1;     /* Power donw analog part of ADC0 */
                          /* MPC5607B/C: saving ~ 2.8mA in STOP */
ADC_1.MCR.B.PWDN = 1;     /* Power down analog part of the ADC1 */
                          /* MPC5607B saving ~ 1.9mA in STOP */
                          /* Please note, not all MPC56xxB derivates */
                          /* support two ADC modules. Also the */
                          /* amount of current might vary as */
                          /* different derivates support a different */
                          /* amount of ADC channels. Status of MDIS, */
                          /* FREEZE, HALT bits of digital */
                          /* peripherals have no impact on STOP */
                          /* power consumption */
                          /* For all peripherals not needed during */
                          /* STOP a Low- Power Peripheral */
                          /* Configuration Register (ME_LP_PC0…7) */
                          /* with STOP = 0 should be assigned to */
                          /* the corresponding Peripheral Control */
                          /* Registers (ME_PCTLx). */
/* Enter STOP Mode */
ME.MCTL.R = ((vuint32_t)STOP_MODE<<28) | 0x00005AF0; /* Mode & Key */
ME.MCTL.R = ((vuint32_t)STOP_MODE<<28) | 0x0000A50F; /* Mode & Key */
while (ME.GS.B.S_MTRANS){}/* Wait STOP mode transition to complete */
}
```

On MPC560xB/C/D derivatives, main VREG is not disabled during STOP or HALT mode if RUN[0..3] mode selects FXOSC to be running and target mode selects FXOSC as system clock. If STOP or HALT is configured with:

- ME_[mode]_MC.MVRON = 0
- ME_[mode]_MC.FIRCON = 0
- ME_[mode]_MC.SYSCLK = 0010/0011, the main VREG will nevertheless remain enabled during the STOP mode if the previous RUN[0..3] mode is configured with ME_RUN[0..3]_MC.FXOSCON = 1. This will result in higher power consumption than expected.

Before entering STOP or HALT mode with the following configuration:

- ME_[mode]_MC.MVRON = 0
- ME_[mode]_MC.FIRCON = 0
- ME_[mode]_MC.SYSCLK = 0010/0011, ensure the RUN[0..3] mode switches off FXOSC and ME_RUN[0..3]_MC.FXOSCON = 0, before attempting to low-power mode transition.

8.3 STANDBY mode entry

STANDBY mode can be requested by SW from DRUN and RUN[0...3] modes.

The lowest power consumption in STANDBY can be achieved by the following sequence:

```
void STANDBY_Mode_Entry(void)
{
    /* Configure STANDBY0 Mode for lowest consumption (only WUP Unit ON) */
    /* Please note, WKPU (Wakeup Unit) is always enabled */
    /* To generate an interrupt event triggered by a wakeup line, it is */
    /* necessary to enable WKPU */
    CAN_0.MCR.B.FRZ = 1;/* Errata e9140PS: */
    CAN_1.MCR.B.FRZ = 1;/* STANDBY can not be entered if the FlexCAN */
    CAN_2.MCR.B.FRZ = 1;/* peripheral is active. If any FlexCAN module */
    CAN_3.MCR.B.FRZ = 1;      /* is enabled at the MC_ME (ME_RUN_PCx/ME_PCTLx) */
    CAN_4.MCR.B.FRZ = 1;      /* and enabled at the FlexCAN module, */
    CAN_5.MCR.B.FRZ = 1;      /* (MCR.B.MDIS=0) STANDBY0 mode will not be */
                              /* entered, and the device will remain in DRUN or */
                              /* RUNx mode. */
                              /* Workaround: The FlexCAN module must be frozen */
                              /* by setting FLEXCANx_MCR[FRZ]=1 in DRUN or */
                              /* RUNx mode before entering STANDBY mode. */
    ME.STANDBY0.B.PDO=1;/* Pad power sequence driver disabled, but state */
                              /* of the output is kept */
    ME.STANDBY0.B.MVRON=0;/* Main Voltage regulator is off */
    ME.STANDBY0.B.DFLAON=1;   /* Data flash is in power-down mode */
    ME.STANDBY0.B.CFLAON=1;/* Code flash is in power-down mode */
    ME.STANDBY0.B.OSCON=0;/* Fast external crystal oscillator (4-16 MHz) off */
    ME.STANDBY0.B.PLLON=0;/* FMPLL is off (default status during STOP0) */
    ME.STANDBY0.B.SYSCLK=0xF; /* System clock disabled BEFORE RC switching off */
    ME.STANDBY0.B.RCON=0;/* Fast internal RC oscillator (16 MHz) off */
    /* Enter STANDBY Mode */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x00005AF0; /* Mode & Key */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x0000A50F; /* Mode & Key */
    while(1){}  /* Wait STANDBY mode transition to complete - use timeout */
}
```

NOTE

- All peripherals need to be configured as off within the ME_PCTLx registers prior to the STANDBY mode entry, otherwise, a reset gets generated on wake-up and the wake-up source is not visible within the WKPU_WISR register.
- Do not deactivate system clock prescaler in CGM_SC_DCx register prior to STANDBY mode entry, otherwise, the system will not be able to exit STANDBY.

9 Low-power wake-up features

There are two mechanisms that can be used for wake up from low-power modes.

- Input pin transition or RTC/API enabled using the WKPU (STANDBY, STOP modes only)
- Peripheral interrupts (HALT, STOP modes only)

The WKPU offers external wake-up/interrupt support through:

- 3 system interrupt vectors for up to 18 interrupt sources for MPC5602/3/4B/C/D
- 3 system interrupt vectors for up to 18 interrupt sources for MPC5602/3/4B/C/D
- 4 system interrupt vectors for up to 24 interrupt sources for MPC5605/6/7B/C/D
- Analog glitch filter per each wake-up line
- Independent interrupt mask
- Edge detection
- Configurable system wake-up triggering from all interrupt sources
- Configurable pullup

The WKPU offers non-maskable interrupt support through:

- Edge detection

- One NMI source with bypassable glitch filter
- Independent interrupt destination: non-maskable interrupt, critical interrupt, or machine check request

On-chip wakeup support is provided through:

- Two wake-up sources (RTC and API)
- Wake-up status mapped to same register as external wake-up/interrupt status

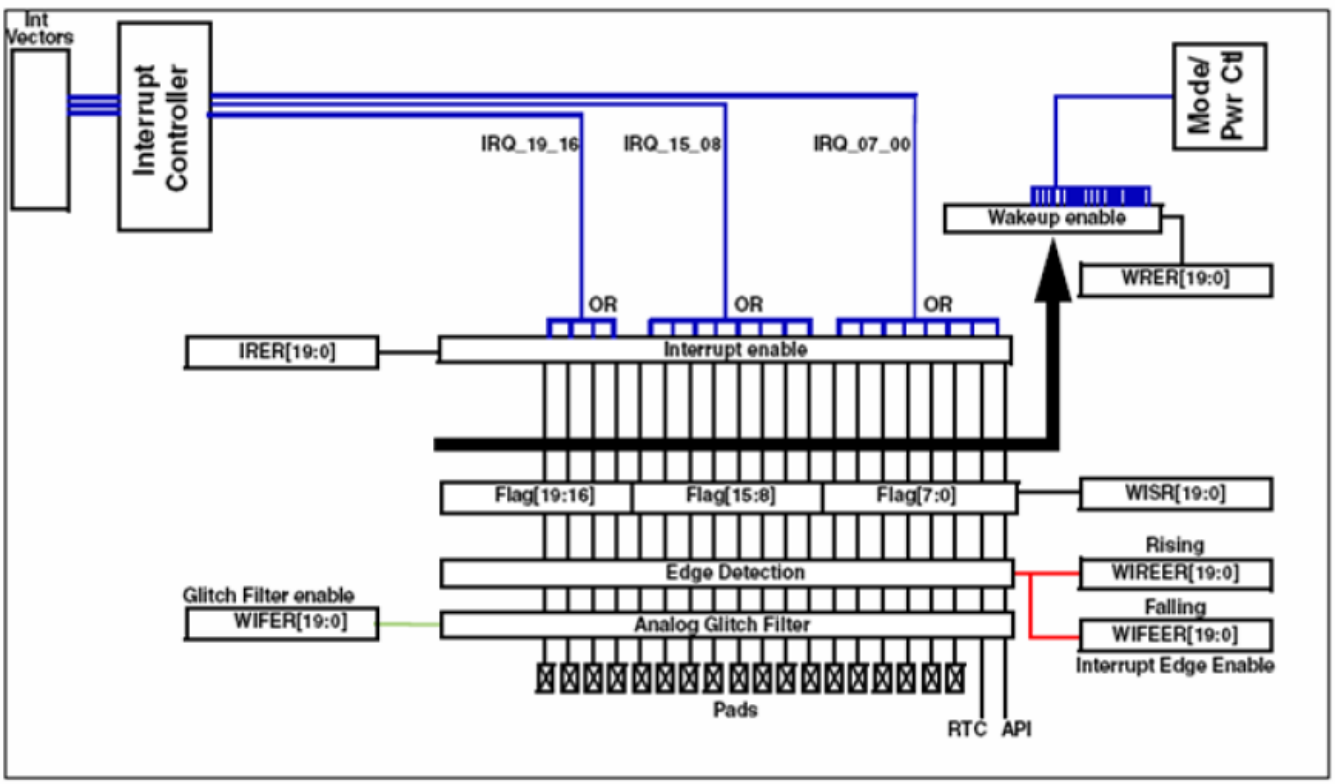


Figure 12. Wake-up unit (WKPU)

NOTE

The WKPU remains powered during low-power modes.

Each WKPU pin:

- Can issue only a wake-up, or an interrupt, or both
- Can be sensitive on rising, falling or both edges
- Has an analog glitch filter, which can be separately enabled
- Has an internal pullup

9.1 Low-power interrupts

The following points must be noted when using interrupts as a wake-up mechanism from low-power modes.

- MSE[EE] status does not affect low-power wake-up interrupts.
- The INTC values do not affect low-power wake-up.
- Only a peripheral's Flag and Interrupt Enable bit are required to notify the ME module to exit STOP or HALT modes.
- If using an interrupt to wake-up from STOP mode, a system clock must be configured in STOP mode. (SYSCLK cannot be 1111)

9.2 MPC56xxB/C/D wake-up line interrupts

	MPC5602/1/D/C	MPC5604/3/2B/C	MPC5607/6/5/B/C
	Interrupt Vector 0	**Interrupt Vector 0**	**Interrupt Vector 0**
WKUP[0]	API	API	API
WKUP[1]	RTC	RTC	RTC
WKUP[2]	PA[1], GPIO[1], E0UC[1], NMI	PA[1], GPIO[1], E0UC[1], NMI	PA[1], GPIO[1], E0UC[1], NMI
WKUP[3]	PA[2], GPIO[2], E0UC[2], MA[2]	PA[2], GPIO[2], E0UC[2]	PA[2], GPIO[2], E0UC[2], MA[2]
WKUP[4]	PB[1], GPIO[17], CAN0RX	PB[1], GPIO[17], CAN0RX	PB[1], GPIO[17], E0UC[31], CAN0RX, LIN0RX
WKUP[5]	PC[11], GPIO[43], MA[2], CAN1RX	PC[11], GPIO[43], CAN1RX, CAN4RX	PC[11], GPIO[43], MA[2], CAN1RX, CAN4RX
WKUP[6]	PE[0], GPIO[64], E0UC[16]	PE[0], GPIO[64], E0UC[16], CAN5RX	PE[0], GPIO[64], E0UC[16], CAN5RX
WKUP[7]	PE[9], GPIO[73], E0UC[23]	PE[9], GPIO[73], E0UC[23], CAN2RX3, CAN3RX	PE[9], GPIO[73], E0UC[23], CAN2RX, CAN3RX
	Interrupt Vector 1	**Interrupt Vector 1**	**Interrupt Vector 1**
WKUP[8]	PB[10], GPIO[26], ADC1_S[6]	PB[10], GPIO[26], ADC0_S[2]	PB[10], GPIO[26], ADC0_S[2], ADC1_S[6]
WKUP[9]	PA[4], GPIO[4], E0UC[4], CS0_1	PA[4], GPIO[4], E0UC[4]	PA[4], GPIO[4], E0UC[4], CS0_1, LIN5RX
WKUP[10]	PA[15], GPIO[15], CS0_0, SCK_0, E0UC[1]	PA[15], GPIO[15], CS0_0, SCK_0	PA[15], GPIO[15], CS0_0, SCK_0, E0UC[1]
WKUP[11]	PB[3], GPIO[19], LIN0RX	PB[3], GPIO[19], SCL, LIN0RX	PB[3], GPIO[19], E0UC[31], SCL, LIN0RX
WKUP[12]	PC[7], GPIO[39], DEBUG[5],LIN1RX	PC[7], GPIO[39], LIN1RX	PC[7], GPIO[39], E1UC[29], DEBUG[5], LIN1RX
WKUP[13]	PC[9], GPIO[41], E0UC[7], DEBUG[7],LIN2RX	PC[9], GPIO[41], LIN2RX	PC[9], GPIO[41], E0UC[7], DEBUG[7], LIN2RX
WKUP[14]	GPIO[75], E0UC[24], CS4_1	PE[11], GPIO[75], CS4_1, LIN3RX	PE[11], GPIO[75], E0UC[24], CS4_1, LIN3RX
WKUP[15]	—	PF[11], GPIO[91]	PF[11], GPIO[91], E1UC[3], CS2_0, LIN4RX
	Interrupt Vector 2	**Interrupt Vector 2**	**Interrupt Vector 2**
WKUP[16]	—	PF[13], GPIO[93], E1UC[26]	PF[13], GPIO[93], E1UC[26], LIN5RX
WKUP[17]	—	PG[3], GPIO[99], E1UC[12]	PG[3], GPIO[99], E1UC[12], CS0_3
WKUP[18]	—	PG[5], GPIO[101], E1UC[14]	PG[5], GPIO[101], E1UC[14], SIN_3
WKUP[19]	PA[0], GPIO[0], E0UC[0], CLKOUT, E0UC[13]	PA[0], GPIO[0], E0UC[0], CLKOUT	PA[0], GPIO[0], E0UC[0], CLKOUT, E0UC[13]
WKUP[20]	—	—	PG[7], GPIO[103], E1UC[16], E1UC[30], LIN6RX
WKUP[21]	—	—	PG[9], GPIO[105], E1UC[18], SCK_2, LIN7RX
WKUP[22]	—	—	PF[9], GPIO[89], E1UC[1], CS5_0, CAN2RX, CAN3RX
WKUP[23]	—	—	PI[3], GPIO[131], E0UC[31], LIN9RX
	Interrupt Vector 3	**Interrupt Vector 3**	**Interrupt Vector 3**
WKUP[24]	—	—	PI[1], GPIO[129], E0UC[29], LIN8RX
WKUP[25]	PB[8], GPIO[24], ADC1_S[4]	—	PB[8], GPIO[24], OSC32K_XTAL, ADC0_S[0], ADC1_S[4]
WKUP[26]	PB[9], GPIO[25], ADC1_S[5]	—	PB[9], GPIO[25], OSC32K_EXTAL, ADC0_S[1], ADC1_S[5]
WKUP[27]	PD[0], GPIO[48], ADC1_P[4]	—	PD[0], GPIO[48], ADC0_P[4], ADC1_P[4]
WKUP[28]	PD[1], GPIO[49], ADC1_P[5]	—	PD[1], GPIO[49], ADC0_P[5], ADC1_P[5]
WKUP[29]	—	—	—
WKUP[30]	—	—	—
WKUP[31]	—	—	—

	Color coding:		
	Not available in 64-pin LQFP	Not available in 64-pin LQFP	—
	—	Not available in 100-pin LQFP and 64-pin LQFP	Not available in 100-pin LQFP
	—	Available only on MPC560xCx and MPC5604BxMG, not available in 64-pin LQFP	—
	—	Available only on MPC560xCx and MPC5604BxMG not on MPC5603BxLx and in 64-	—
	—	—	Not available in 100LQFP and 144-pin LQFP
	—	—	This wakeup lines cannot exit STANDBY mode

Figure 13. MPC56xxB/C/D wake-up line interrupts

Package	MPC5602/1/D/C	MPC5604/3/2B/C	MPC5607/6/5/B/C
64-pin LQFP	14 wakeup sources (2 internal, 12 external)	12 wakeup sources (2 internal, 10 external)	—
	API	API	
	RTC	RTC	
	WKUP[2:4]	WKUP[2:4]	
	WKUP[8:13]	WKUP[8:13]	
	WKUP[19]	WKUP[19]	
	WKUP[25:26]		
100-pin LQFP	20 wakeup sources (2 internal, 18 external)	16 wakeup sources (2 internal, 14 external)	18 wakeup sources (2 internal, 16 external)
	API	API	API
	RTC	RTC	RTC
	WKUP[2:14]	WKUP[2:14]	WKUP[2:14]
	WKUP[19]	WKUP[19]	WKUP[19]
	WKUP[25:28]		WKUP[25:28]
144-pin LQFP	—	20 wakeup sources (2 internal, 18 external)	27 wakeup sources (2 internal, 25 external)
		API	API
		RTC	RTC
		WKUP[2:19]	WKUP[2:22]
			WKUP[25:28]
176-pin LQFP	—	—	29 wakeup sources (2 internal, 27 external)
			API
			RTC
			WKUP[2:28]

Figure 14. Overview packages versus wake-up pins

Wakeup Number	Port	SIU PCR #	Wakeup IRQ to INTC	WIPUER Pull Enable Bits	WISR Int Status Flags	WISR Flag Clearing Mask	Register Bit Position
WKUP0	API	n.a.	WakeUp_IRQ_0	n.a.	n.a.	0x00000001	31
WKUP1	RTC	n.a.	WakeUp_IRQ_0	n.a.	n.a.	0x00000002	30
WKUP2	PA1	PCR1	WakeUp_IRQ_0	IPUE2	EIF2	0x00000004	29
WKUP3	PA2	PCR2	WakeUp_IRQ_0	IPUE3	EIF3	0x00000008	28
WKUP4	PB1	PCR17	WakeUp_IRQ_0	IPUE4	EIF4	0x00000010	27
WKUP5	PC11	PCR43	WakeUp_IRQ_0	IPUE5	EIF5	0x00000020	26
WKUP6	PE0	PCR64	WakeUp_IRQ_0	IPUE6	EIF6	0x00000040	25
WKUP7	PE9	PCR73	WakeUp_IRQ_0	IPUE7	EIF7	0x00000080	24
WKUP8	PB10	PCR26	WakeUp_IRQ_1	IPUE8	EIF8	0x00000100	23
WKUP9	PA4	PCR4	WakeUp_IRQ_1	IPUE9	EIF9	0x00000200	22
WKUP10	PA15	PCR15	WakeUp_IRQ_1	IPUE10	EIF10	0x00000400	21
WKUP11	PB3	PCR19	WakeUp_IRQ_1	IPUE11	EIF11	0x00000800	20
WKUP12	PC7	PCR39	WakeUp_IRQ_1	IPUE12	EIF12	0x00001000	19
WKUP13	PC9	PCR41	WakeUp_IRQ_1	IPUE13	EIF13	0x00002000	18
WKUP14	PE11	PCR75	WakeUp_IRQ_1	IPUE14	EIF14	0x00004000	17
WKUP15	PF11	PCR91	WakeUp_IRQ_1	IPUE15	EIF15	0x00008000	16
WKUP16	PF13	PCR93	WakeUp_IRQ_2	IPUE16	EIF16	0x00010000	15
WKUP17	PG3	PCR99	WakeUp_IRQ_2	IPUE17	EIF17	0x00020000	14
WKUP18	PG5	PCR101	WakeUp_IRQ_2	IPUE18	EIF18	0x00040000	13
WKUP19	PA0	PCR0	WakeUp_IRQ_2	IPUE19	EIF19	0x00080000	12
WKUP20	PG7	PCR103	WakeUp_IRQ_2	IPUE20	EIF20	0x00100000	11
WKUP21	PG9	PCR105	WakeUp_IRQ_2	IPUE21	EIF21	0x00200000	10
WKUP22	PF9	PCR89	WakeUp_IRQ_2	IPUE22	EIF22	0x00400000	9
WKUP23	PI3	PCR131	WakeUp_IRQ_2	IPUE23	EIF23	0x00800000	8
WKUP24	PI1	PCR129	WakeUp_IRQ_3	IPUE24	EIF25	0x01000000	7
WKUP25	PB8	PCR24	WakeUp_IRQ_3	IPUE25	EIF25	0x02000000	6
WKUP26	PB9	PCR25	WakeUp_IRQ_3	IPUE26	EIF26	0x04000000	5
WKUP27	PD0	PCR48	WakeUp_IRQ_3	IPUE27	EIF27	0x08000000	4
WKUP28	PD1	PCR49	WakeUp_IRQ_3	IPUE28	EIF28	0x10000000	3

bonded in 100LQFP, 144LQFP, 176LQFP and 208MAPBGA
bonded in 144LQFP,176LQFP and 208MAPBGA
bonded in 176LQFP and 208MAPBGA

Figure 15. MPC5607B wake-up

NOTE

Small packages do not contain all ports.

9.3 Wakeup registers

Name	Symbol	Description
Wakeup/Interrupt Status Flag Register	WISR	Flags event as defined by WIREER and WIFEER
Wakeup/Interrupt Rising-Edge Event Enable Register	WIREER	Enables rising-edge event
Wakeup/ Interrupt Falling-Edge Event Enable Register	WIFEER	Enables rising-edge event
Interrupt Request Enable Register	IRER	Enables flags to cause interrupt request
Wakeup/Interrupt Filter Enable Register	WIFER	Enables analog glitch filter on external pad input (filter glitch < 40 ns, passes signals > 1000 ns)
Wakeup/Interrupt Pullup Enable Register	WIPUER	Enables pullup on external pad (use for all pads to minimize leakage)
NMI Configuration Register	NCR	Configuration settings for NMI
NMI Status Flag Register	NSR	Holds NMI Status flags

9.4 Low-power timer wake-ups

Often a system may have to restart on a regular basis. This may involve the maintenance of an external system that needs a regular check, or service.

For such use cases, an energy-efficient solution can be realized using the Real Time Counter (RTC) / Autonomous Periodic Interrupt (API) feature which has been implemented on the MPC56xx family.

The RTC is a free-running counter used for timekeeping applications.

- Generates interrupt independent of RUN or low-power mode
- First generates a wake-up for low-power exit, and then interrupt, if required
- Continues counting through resets except for power-up reset
- Three selectable counter clock sources with optional prescalers
    - SIRC (128 kHz)
    - SXOSC (32 kHz)
    - FIRC (16 MHz)
- API provides regular timeouts for wake-up / interrupt.
- Compares lower 10 bits of RTC to 10-bit compare value
- At match, automatically adds programmed value for next compare
- RTC provides longer timeout for wake-up / interrupt
- Compares 12 bits above API to 12-bit compare value

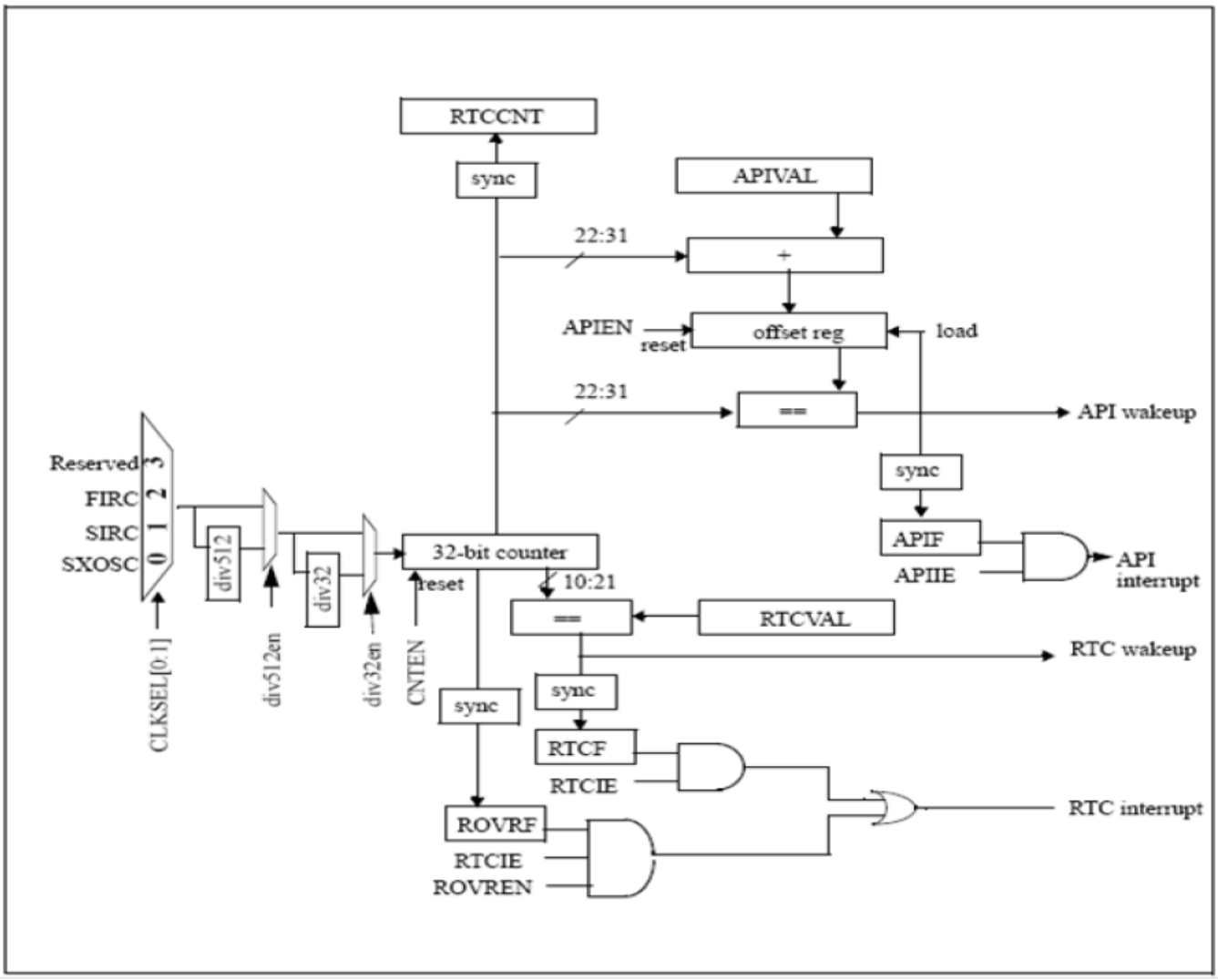


Figure 16. RTC / API block diagram

NOTE

32 kHz SXOSC option is not supported on all derivates.

As can be seen in Figure 16, the RTC clock source is selectable from the 16 MHz FIRC, 32 kHz SIRC, or 32 kHz SXOSC. RTC and API share a common 32-bit, free-running counter.

Table 13 , which has originally been published in the application note AN2865: Qorivva Simple Cookbook, shows RTC/API timeouts based on different clock source and divider settings.

Table 13. RTC/API timeouts

Clock source	div512 RTC_RTCC [div512]	div 32 RTC_RTCC [div 32]	RTC Counter Input Clock Frequency	RTC Counter Inout Clock Period (1/ (RTC Counter Input Clock Freq.))	Min. API Timeout (1 x RTC Counter input clock period)	Max. API Timeout ((2^10 - 1) x RTC Counter input clock period	Min. RTC Timeout (2^10 x RTC Counter input clock period)	Max. RTC Timeout ((2^22 - 1) x RTC Counter input clock period)	RTC Rollover Timeout (2^32 x RTC Counter input clock period)
128 kHz SIRC[1] with SIRCDIV= 0	0	0	128 kHz	~ 7.8 µs	~ 7.8 µs	~ 8 ms	~ 8 ms	~ 31 s	~ 8.9 h
	0	1	4 kHz	250 µs	250 µs	~ 256 ms	256 ms	~ 17 min	~ 12 d
	1	0	250 Hz	4 ms	4 ms	~ 4.09 s	~ 4.10 s	~ 4.4 h	~ 6.3 mo.
	1	1	7.8125 Hz	128 ms	128 ms	~131 s	~ 131 s	~ 6 d	~ 17 yr.
16 MHz FIRC[2] (125 x SIRC)	0	0	16 MHz	62.5 ns	62.5 ns	~ 64 µs	64 µs	0.25 s	~ 4.3 min
	0	1	500 kHz	2 µs	2 µs	~ 2 ms	2.048 ms	8 s	~ 2.8 h
	1	0	31.25 kHz	32 µs	32 µs	~ 33 ms	~ 33 ms	~ 2.1 min	~ 1.5 d
	1	1	~977 Hz	1.024 ms	1.024 ms	~ 1048 s	~ 1049 s	~ 1.1 h	~ 1.6 mo.
32 kHz SXOSC[3] or 128 kHz SIRC/4[1]	0	0	32 kHz	31.25 µs	31.25 µs	~32 ms	32 ms	~ 2.1 min	~ 1.5 d
	0	1	1 kHz	1 ms	1 ms	1.023 s	1.024 s	~ 1.1 h	~ 1.6 mo.
	1	0	62.5 Hz	16 ms	16 ms	~ 16 s	~ 16 s	~ 18 h	~ 2.1 yr.
	1	1	~2 Hz	512 ms	512 ms	~ 524 s	~524 s	~ 24 d	~ 67 yr.
FXOSC[4]	0	0	8 MHz	125 ns	125 ns	~ 128 s	~ 128 s	0.5 s	~ 8.6 min
	0	1	250 kHz	4 µs	4 µs	~ 4.1 ms	~ 4.1 ms	16 s	~ 4.6 h
	1	0	15.625 kHz	64 µs	64 µs	~ 65 ms	66 ms	~ 4.3 min	~ 3 d
	1	1	~ 488 Hz	2.048 ms	2.048 ms	~ 2.1 s	~ 2.1 s	~ 2.3 h	~ 3.2 mo.

1. SIRC is divided by 4 to 32 kHz using reset default value in CGM_SIRC_CTL[SIRCDIV] for MPC56xxB, or GGM_LPRC_CTL[LPRCDIV] for MPC56xxS
2. FIRC is divided by 1 using reset default value in CGM_RC_CTL[RCDIV] for MPC56xxB, MPC56xxS
3. 32 kHz SXOSC not available in STANDBY mode
4. FXOSC as RTC/API option avaialble only in MPC56xxS. Also, not available in STANDBY mode

In choosing to use the RTC or the API, the user needs to trade off the length of time the timer may have to wait before wake-up against resolution.

NOTE

Although an API timeout will consistently trigger at identical periods, the first API timeout can't be predicted. Since there is internal synchronization of clocks occurring when APIVAL is written, an additional time in the region of two clock cycles will occur. This depends on the clocks being used and the divider values. Whilst it is not determinable in advance, it will always be a consistent amount and would not vary in an application once set.

Offset: 0x4 | Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	CNTEN	RTCIE	FRZEN	ROVREN	RTCVAL											
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	APIEN	APIIE	CLKSEL		DIV512EN	DIV32EN	APIVAL									
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 17. RTC Control (RTCC) register

NOTE

The RTC needs to be disabled by setting CNTEN = 0 prior to any update of the RTCVAL field.

Table 14. RTCC field descriptions

Field	Description
CNTEN	Counter Enable CNTEN enables the RTC counter. Setting CNTEN to 1'b0 has the effect of asynchronously resetting (synchronous reset negation) all the RTC and API logic. This allows for the RTC configuration and clock source selection to be updated without causing synchronization issues. 1 Counter enabled 0 Counter disabled
RTCIE	RTC Interrupt Enable The RTCIE field enables interrupts requests to the system if RTCF is asserted. 1 RTC interrupts enabled 0 RTC interrupts disabled
FRZEN	Freeze Enable The counter freezes on entering the Debug mode on the last valid count value if FRZEN is set. After coming out of the Debug mode, the counter starts from the frozen value. 0 Counter does not freeze in Debug mode. 1 Counter freezes in Debug mode.
ROVREN	Counter Roll Over Wakeup/Interrupt Enable ROVREN enables wake-up and interrupt requests when the RTC has rolled over from

Table continues on the next page...

Table 14. RTCC field descriptions (continued)

Field	Description
	0xFFFF_FFFF to 0x0000_0000. RTCIE must also be set in order to generate an interrupt from a counter rollover. 1 RTC rollover wake-up/interrupt is enabled. 0 RTC rollover wake-up/interrupt is disabled.
4:15 RTCVAL[0:11]	RTC Compare Value The bits of this field are compared to bits 10–21 of the RTC counter and if these match, RTCF is set. RTCVAL can be updated when the counter is running.
APIEN	Autonomous Periodic Interrupt Enable APIEN enables the autonomous periodic interrupt function. 1 API enabled 0 API disabled
APIIE	API Interrupt Enable The APIIE field enables interrupts requests to the system if APIF is asserted. 1 API interrupts are enabled. 0 API interrupts are disabled.
18:19 CLKSEL[0:1]	Clock Select This field selects the clock source for the RTC. CLKSEL may only be updated when CNTEN is 0. The user must ensure that oscillator is enabled before selecting it as a clock source for RTC. 00 SXOSC 01 SIRC 10 FIRC 11 Reserved
DIV512EN	Divide by 512 Enable DIV512EN enables the 512 clock divider. DIV512EN may only be updated when CNTEN is 0. 0 Divide by 512 is disabled. 1 Divide by 512 is enabled.
DIV32EN	Divide by 32 Enable DIV32EN enables the 32 clock divider. DIV32EN may only be updated when CNTEN is 0. 0 Divide by 32 is disabled. 1 Divide by 32 is enabled.
22:31 APIVAL[0:9]	API Compare Value The bits of this field are compared with bits 22–31 of the RTC counter, and if match occurs, an

Table 14. RTCC field descriptions

Field	Description
	interrupt/wake-up request is asserted. APIVAL may only be updated when APIEN is 0, or API function is undefined. **NOTE:** API functionality starts only when APIVAL is nonzero. The first API interrupt takes two more cycles because of synchronization of APIVAL to the RTC clock. After that, interrupts are periodic in nature. The minimum supported value of APIVAL is 4.

There are separate fields within the RTCC register to control the operation and parameters for the RTC and API systems. The clock source select bits and initial values are all assignable as is the rollover enable for the RTC and the individual enable bits for the counters. Bits are available to enable interrupts for each of the mechanisms as well as for the divider options.

As a reference clock is always required to feed the RTC/API counter to enable wake-ups, it needs to be ensured that the selected clock source for the RTC module is running while in LPM. For LPM periodic wake-up applications, it is recommended to use the Slow Internal RC Oscillator (SIRC). This is because it offers the lowest possible power consumption. However, the SIRC has a tolerance of ±10%. For conditions and details, see the latest data sheet of the device being used, available on http://www.freescale.com.

By default, the SIRC is enabled in all RUN and low-power modes except STANDBY mode. To enable SIRC in STANDBY, the Low Power RC Control Register (SIRC_CTL) needs to be configured accordingly by setting SIRC control in STANDBY mode to 1, or SIRC_CTL[SIRCON_STDBY] = 1.

The next lowest power option is using the 32 kHz external oscillator facility available on some MPC56xxB derivatives.

The 16 MHz IRC is the most accurate internal clock available (<5%), consuming a higher current but featuring a fast startup time of typical 1.1 µs. This capability allows registers, and modules, to be configured while waiting for the external crystal to stabilize. Alternatively, it supports rapid code execution during short run cycles, enabling the device to return to an LPM much more quickly.

To wake the device from LPM via RTC/API, the external wake-up within the Wakeup Request Enable Register (WKPU_WRER) as well as the rising-edge of WUP0 within Wakeup/Interrupt Rising-Edge Event Enable Register (WKPU_WIREER) needs to be enabled. The RTC/API generates only wake-up events when configured to the rising-edge.

The software example given below shows a possible RTC initialization:

```
void Init_RTC_Wakeup(void)
{   /* RTC WakeUp line Mgm - WKUP0 */
    WKUP.WRER.R = 0x42;;      /* Enable WUP0 */
    WKUP.WIREER.R = 0x42;    /* Enable Rising Edge of WUP0 */
    WKUP.WISR.R = 0xFFFFF;
    CGM.LPRC_CTL.R = 0x301;;
    RTC.RTCC.R = 0;
    RTC.RTCC.R = 0xA01B1000;
    WKUP.WISR.R = 2;
    Configure_RUN1_Mode();
    /* Enter in RUN1 Mode */
    ME.MCTL.R = ((vuint32_t)RUN1_MODE<<28) | 0x00005AF0;     /* Mode & Key */
    ME.MCTL.R = ((vuint32_t)RUN1_MODE<<28) | 0x0000A50F;     /* Mode & Key */
    while (ME.IS.B.MTC != 1) {}  /* Wait Until transition completed */
    ME.IS.B.MTC = 1;         /* Clear bit */
}
```

To realize a periodic wake-up scenario, the following sequence could be used after the RTC wake-up/ timeout occurred.

```
if(RTC.RTCS.B.RTCF == 1)
{
      RTC.RTCC.R =0;
      RTC.RTCC.R =0xA01B1000;
      RTC.RTCS.B.RTCF =1;
      WKUP.WISR.R = 0x00000002;
}    /* Now, RTC is setup again to allow the next low power mode entry */
```

9.5 External pin wake-up

The WKPU has several signal inputs that can be used as external interrupt sources in normal RUN mode or as system wake-up sources in all power-down modes.

These external signal inputs include one signal input that can be used as a non-maskable interrupt source in normal RUN, HALT or STOP modes or a system wake-up source in STOP or STANDBY modes. On MPC5607B/C, there is NMI pin configuration limitation in STANDBY mode. The NMI pin cannot be configured to generate non-maskable interrupt event to the core (WKPU_NCR[NDSS] = 00), if the following STANDBY mode is to be used:

- NMI pin enabled for wake-up event
- Exit sequence boot from RAM
- Code flash module power-down on STANDBY exit sequence

With the configuration given above, the following scenario may occur:

- System is in standby.
- NMI event is triggered on PA[1].
- System wakes up z0 core power domain.
- z0 core reset is released and NMI event is sampled by core on the first clock-edge. The z0 core attempts to fetch code at 0x10 address (IVPR is not yet initialized by application) and receives an exception since the flash is not available.

If NMI is configured as wake-up source, WKPU_NCR[NDSS] must be configured as 11. This will ensure no NMI event is triggered on the core but system wake-up is triggered. After STANDBY exit, core will boot and configure its IVOR/IVPR. It may then reconfigure WKPU_NCR:DSS to the appropriate configuration for enabling NMI/CI/MCP.

NOTE

The user must be aware that the wake-up pins are enabled in all the modes, therefore, the wake-up pins must be correctly terminated to ensure minimal current consumption. Any unused wake-up signal input must be terminated using an external pullup or pulldown, or by internal pullup enabled at WKUP_WIPUER. Also, care has to be taken on packages where the wake-up signal inputs are not bonded. For these packages, the user must ensure the internal pullups are enabled for the signals which are not bonded.

```
void Init_Wake_Up_Inputs(void)
{
    SIU.PCR[64].R = 0x103; /* Switch 1 wake-up input - PE[0] =
WKUP[6] */
    WKUP.WIPUER.R = 0x0003ffff;         /* Enable pullups on all
18 wakeup pins */
    WKUP.WISR.R = 0x00000008;           /* Clear WKUP[6]flag */
    WKUP.WRER.R = 0x00000008;           /* Enable WKUP[6]for
wakeup event */
    WKUP.WIREER.R = 0x00000008;         /* Enable Rising Edge of
WKUP[6] */
  /*WKUP.WIFEER.R = 0x00000008;      /* Enable Falling Edge of
WKUP[6] */
    WKUP.WIFER.R = 0x00000008;          /* Enable Filter of
WKUP[6] */
}
```

9.6 Wake-up source determination

In order to allow software to determine the wake-up source at one location, on-chip wake-ups are reported along with external wake-ups in the WISR register. Enabling and clearing of these wake-ups are done via the on-chip wake-up source's own registers. Each external interrupt supports an individual flag which is held in the flag register (WISR). This register is a clear-by-write-1 register type, preventing inadvertent overwriting of other flags in the same register.

9.7 Wake-up interrupt filter

The Wakeup/Interrupt Filter Enable Register (WKPU_WIFER) is used to enable an analog filter on the corresponding interrupt pads to filter out glitches on the inputs. The number of wake-ups/interrupts supporting this feature is documented in the relevant reference manual of the specific device of the MPC56xx family, available on http://www.freescale.com, as it varies across different devices of the MPC56xx family.

NOTE

There is no analog filter for the RTC/API interrupts.

.

10 Low-power mode exit

Exit from low-power modes is caused by:

- System reset into DRUN mode
- Hardware failure into SAFE mode
- Predefined interrupt event
- Predefined wake-up event

Table 15. Low-Power mode exit

	WAIT	HALT	STOP	STANDBY
Into DRUN via system reset	Yes	Yes	Yes	Yes
Into SAFE on HW failure	Yes	Yes	Yes	No
Into RUN0...3 via interrupt	Yes	Yes	Yes	No
Into RUN0...3 via wakeup	No	No	Yes	No
Into DRUN via wakeup	No	No	No	Yes

10.1 Restart after WAIT instruction

Once the WAIT instruction has been executed, the system supports very short restart latency of less than four system clocks. If enabled, asynchronous interrupts which cause the idle mode entered by WAIT instruction to be exited are critical input, external input, and machine check. Non-maskable interrupts also cause the waiting state to be exited.

10.2 HALT mode exit

The system can exit from the HALT mode by:

- Interrupt
- Pin transition
- RTC/API timeout

10.3 STOP mode exit

Upon exit from STOP mode by pin transition, RTC/API event, or interrupt, only FIRC can be the system clock.

On MPC5607/6/5B/C and MPC5604/3/2B/C, a software timer (SWT) interrupt does not cause STOP mode exit. While in STOP mode, if the SWT is configured to generate an interrupt and the system clock is disabled (ME_STOP_MC[SYSCLK] = 0xF), a SWT interrupt event will not trigger an exit from STOP mode.

Other internal or external wake-up events such as RTC, API, or WKPU pins, are not affected and will trigger a STOP exit independent of the ME_STOP_MC[SYSCLK] configuration.

If a SWT interrupt is to be used to wake the device during STOP mode, software may not disable the system clock (ME_STOP_MC[SYSCLK] = 0xF).

In RUNx mode after STOP mode exit, the system RAM can be accessed before it is ready. This is valid only for MPC5604/3/2B/C and if ME_STOP_MC[FXOSC] is enabled, ME_STOP_MC[FIRC] is disabled, ME_RUNx_MC[FIRC] is enabled and ME_RUNx_MC[SYSCLK] = FXOSC or FXOSC_DIV.

At the transition of STOP to RUNx, the RUNx mode can be entered before the system RAM is ready. If the application software accesses the RAM during this time, the RAM value cannot be determined.

There are two workarounds possible:

- Do not disable the IRC, if the system clock source is not disabled.

  The XOSC draws a lot more current than the IRC, so there should be no noticeable increase in the STOP mode power consumption.

- Have the software check that the mode transition has completed via the ME_GS register before accessing the system RAM.

10.4 STANDBY mode exit

When exiting STANDBY mode by pin transition or timer wake-up, there are two options for code execution depending on the configuration of RGM_STDBYBOOT_FROM_BKP_RAM:

- System boots from flash on STANDBY exit (reset default)
- System boots from backup RAM on STANDBY exit

Configuring system boot from backup RAM has two advantages:

- Enables lower power consumption because the flash does not need to be fully powered. Booting from SRAM and writing C code needs an initialization of the stack pointer and small data areas, which is part of a normal compiler initialization on reset. However, a smaller code can be written in assembly and has a shorter boot sequence, if booting from SRAM.
- Faster startup time because the flash does not need to be power sequenced.

The amount of power savings when booting from RAM and STANDBY wake-up varies by application.

NOTE

On exit from STANDBY mode, most of the registers are reset as in a power-up reset. Power domain 0 registers are not reset, including RGM registers, ME _DRUN_MC register and PCU registers, RTC registers, CAN SAMPLER registers, and so on.

10.4.1 Boot from SRAM after STANDBY exit

The following things must be considered while booting from SRAM after STANDBY exit.

- Make sure the linker file supports code placement in SRAM.
- Allow “far calls” by selecting the corresponding compiler option.
- Once executing code out of SRAM after STANDBY exit, some CPU registers need to be initialized:
  - r1: Initialize stack pointer
  - r13: Initialize r13 to rw SDA base
  - r2: Initialize r2 to ro SDA base
  - r1: Terminate stack

- Call: SRAM_Boot_Function

```
void Enter_STANDBY_Exit_Into_SRAM(void)
{
    /* Configure GPIO[3] */
    SIU.PCR[3].R=0x0200;
    SIU.GPDO[3].B.PDO = 0;

    /* Configure GPIO[4] */
    SIU.PCR[4].R=0x0200;
    SIU.GPDO[4].B.PDO = 0;

    /*Configure WUP3 to exit from STANDBY */
    WKUP.WIPUER.R = 0x0003ffff;  /* Enable pullups on all 18 wakeup pins (even if by
default they are already pulluped) */
    WKUP.WISR.R = 0x00000008;    /* Clear WUP3 flag */
    WKUP.WRER.R = 0x00000008;    /* Enable WUP3 for wakeup event */
    /*WKUP.WIREER.R = 0x00000008;  /* Enable Rising Edge of WUP3 */
    WKUP.WIFEER.R = 0x00000008;  /* Enable falling Edge of WUP3 */
    WKUP.WIFER.R = 0x00000008;   /* Enable Filter of WUP3 */

    ME.RUNPC[1].R = 0x00000028;/* Peripherals enabled only in DRUN,RUN1 Mode */
    ME.LPPC[0].R =  0;/* Peripherals disabled in STOP/HALT/STANDBY Mode */
    ME.LPPC[1].R =  0x00002000;/* periphs enabled only in STANDBY Mode */
    /*ME.PCTL[72].R = 0x09;/* select the RUNPC1,LPPC1 for EMIOS0 */

    Configure_STANDBY0_Mode_SRAM_Exit();

    /* Enter in RUN1 Mode */
    ME.MCTL.R = ((vuint32_t)RUN1_MODE<<28) | 0x00005AF0;     /* Mode & Key */
    ME.MCTL.R = ((vuint32_t)RUN1_MODE<<28) | 0x0000A50F;     /* Mode & Key */
while (ME.IS.B.MTC != 1) {}     /* Wait Until transition completed */
    ME.IS.B.MTC = 1;     /* Clear bit */

    /*Enter in STANDBY Mode */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x00005AF0;  /* Mode & Key */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x0000A50F;  /* Mode & Key */

    /* Call function located in RAM */
}



#include "jdp.h"
//#include "test.h"
//#include "project.h"

//#define FLASH_PD
```

Low-power mode exit

```
//#define FLASH_LP
//#define STDBY_RAM
#define EXIT_INTO_SRAM
#pragma ghs section rodata ".RAM.code"

//#pragma ghs section text = ".code"

void SRAM_Boot_Function(void)
{
#ifdef FOJDHSDF
    Disable_Watchdog();

    #ifdef FLASH_PD

    //Flash in Power Down
    ME.DRUN.B.CFLAON=1;
    ME.DRUN.B.DFLAON=1;

    //re-enter in DRUN mode to switch-off the flash
    ME.MCTL.R = 0x30005AF0;       // Mode & Key
    ME.MCTL.R = 0x3000A50F;       // Mode & Key
    while (ME.IS.B.MTC != 1) {}  // Wait Until transition completed
    ME.IS.B.MTC = 1;  // Clear bit

    while(SIU.GPDI[2].R);  // wait pushing GPIO[38] to go ahead

    //enter in RUN0 mode to switch-on the flash
    ME.MCTL.R = 0x40005AF0;       // Mode & Key
    ME.MCTL.R = 0x4000A50F;       // Mode & Key
    while (ME.IS.B.MTC != 1) {}  // Wait Until transition completed
    ME.IS.B.MTC = 1;  // Clear bit

    #endif

    #ifdef FLASH_LP
    //Flash in Low Power in DRUN Mode
ME.DRUN.B.CFLAON=2;
    ME.DRUN.B.DFLAON=2;

    //re-enter in DRUN mode to update the config
    ME.MCTL.R = 0x30005AF0;       // Mode & Key
    ME.MCTL.R = 0x3000A50F;       // Mode & Key
    #endif

    #ifdef EXIT_INTO_SRAM
    //WakeUp Mgm - Wakeup_IRQ_0  - WKUP3
    //(GPIO2 - PA[2] - J16.2) configured as input by default

    //Configure WUP3 to exit from STANDBY
    WKUP.WRER.R = 0x00000008;    //Enable WUP3 for wakeup event
    WKUP.WIREER.R = 0x00000008;  //Enable Rising Edge of WUP3
    WKUP.WIFER.R = 0x00000008;   //Enable Filter of WUP3

    //Configure GPIO[1]
    SIU.PCR[1].R=0x0200;
    SIU.GPDO[1].B.PDO = 0;

    //Flash in Low Power in DRUN Mode - Commented otherwise crash!
    //ME.DRUN.B.CFLAON=1;
    //ME.DRUN.B.DFLAON=1;

    //Exit from STANDBY Mode on backup RAM
    RGM.STDBY.B.BOOT=1;/* Exit from STANDBY Mode on backup RAM */
    RGM.STDBY.B.DRUND_FLA=0;/* Data Flash in Low Power Mode */
    RGM.STDBY.B.DRUNC_FLA=0;/* Data Flash in Low Power Mode */


    /* Enter STANDBY Mode from RUN1 Mode */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x00005AF0;  /* Mode & Key */
    ME.MCTL.R = ((vuint32_t)STANDBY_MODE<<28) | 0x0000A50F;  /* Mode & Key */
```

```
#endif
#endif
}

#pragma ghs section text=default
```

10.5 Wake-up timing

If the main regulator is also off in device low-power modes, then during the exit sequence, the flash is kept in its low-power state and is switched on only when the Main Voltage Regulator Switch-On process has completed.

NOTE

All exit times quoted in the forthcoming sections are from single examples, and must be regarded as typical room temperature values. The time duration may vary slightly from device to device.

10.5.1 Restart after WAIT instruction

WAIT mode supports very short restart latency of less than four system clocks.

10.5.2 HALT mode

10.5.2.1 Halt mode configuration

The following table depicts an example of a typical HALT mode configuration.

Table 16. Typical HALT mode configuration

System clock	8 KB RAM option	32 KB RAM option	CFLASH	FMPLL	FXOSC	SIRC	FIRC	FIRC DIV
16 MHz	Enabled	Disabled	Low Power	Power Down	Power Down	OFF	ON	OFF

10.5.2.2 Typical HALT mode wake-up timing

The table below shows typical timing for HALT mode wake-up.

Table 17. Typical HALT mode wake-up timing

	HW wake-Up sequence	HW Wake-Up Sequence[1]	CFLASH & DFLASH RESET exit	FXOSC startup	FMPLL lock time	ADC power-up
MPC56xx	12 µs	0.5 µs	0 µs	See Note[1]	40 µs	1.5 µs

1. For this HALT wake-up example, the HW wake-up sequence contains the system initialization time.

10.5.3 STOP mode

10.5.3.1 STOP mode configuration

The table below shows a typical STOP mode configuration.

Table 18. Typical STOP mode configuration

System Clock	8 KB RAM Option	32 KB RAM option	CFLASH	FMPLL	FXOSC	SIRC	FIRC	FIRC DIV
Disabled	Enabled	Disabled	Power Down	Power Down	Power Down	OFF	OFF	OFF

10.5.3.2 Typical STOP mode wake-up timing

In the following example for STOP mode exit, the HW wake-up sequence is essentially:

- The time taken for the ULPV (Ultra Low Power Regulator) to stabilize and IRC startup (10 µs)
- The main VREG stabilization time (12 µs)
- The time taken for the reset state machine to complete (20 µs)

The CFLASH and DFLASH exit times occur in parallel with the HW wake-up reset sequence, so the user can execute code from flash after the 42 µs sequence.

The following table shows the typical STOP mode wake-up timing.

Table 19. Typical timing for STOP mode wake-up

	HW wake-up sequence	CFLASH & DFLASH Low Power exit	CFLASH & DFLASH RESET exit	FXOSC startup	FMPLL lock time	ADC power-up
MPC56xx	42 µs	34 µs	30 µs	See Note[1]	40 µs	1.5 µs

1. For this STOP wake-up example, the HW wake-up sequence contains: ULPV stabilisation, FIRC Start-Up, main VREG stabilisation and system initialization time.

The plot given in Figure 18 shows a measured exit from STOP mode. The blue line shows a wakeup pin transitioning to a low state and triggering the wakeup sequence. The yellow line shows an output pin toggling high as caused by the first line of code executed after STOP recovery.

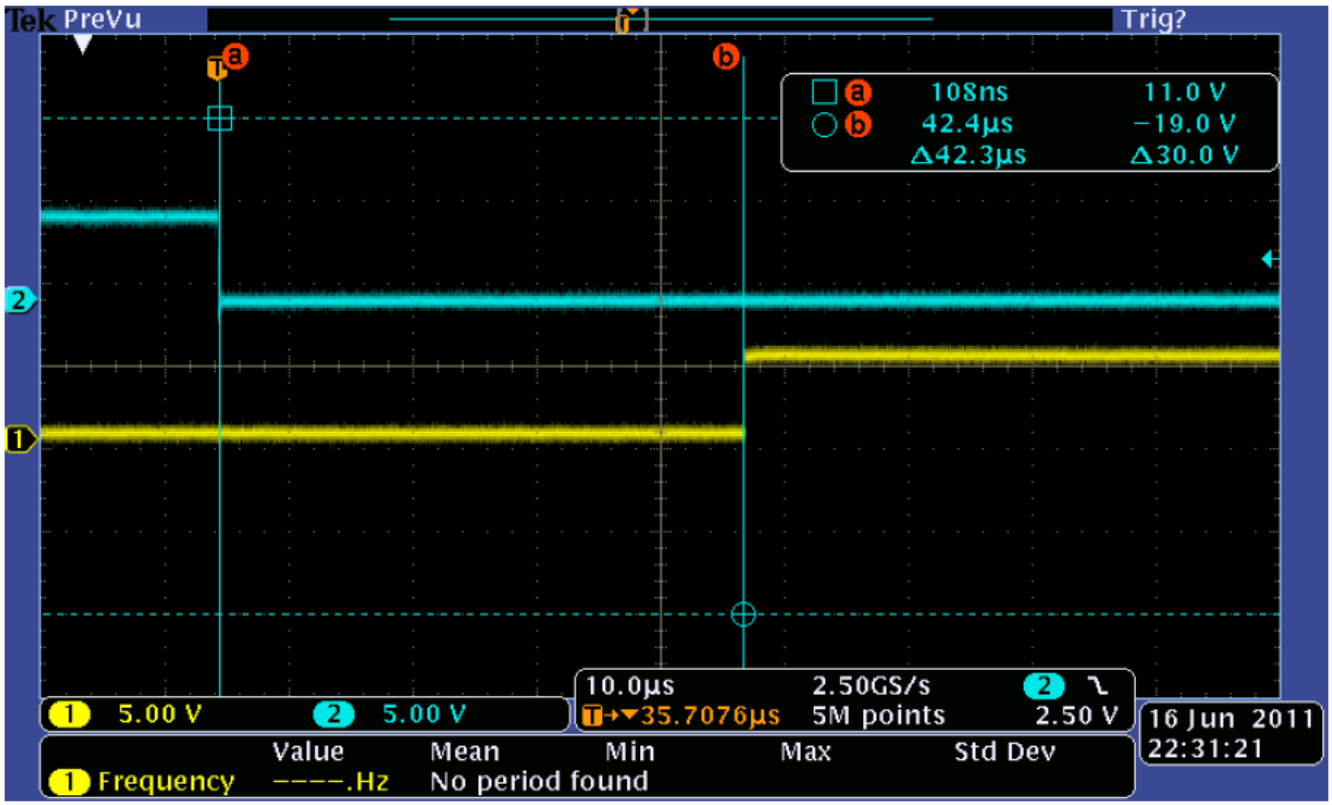


Figure 18. STOP mode exit

10.5.4 STANDBY mode

10.5.4.1 STANDBY mode configuration

The following table shows the example of a typical STANDBY mode configuration.

Table 20. STANDBY mode configuration

System clock	8 KB RAM option	32 KB RAM option	CFLASH	FMPLL	FXOSC	SIRC	FIRC	FIRC DIV
Disabled	Enabled	Disabled	Power Down	Power Down	Power Down	OFF	OFF	OFF

10.5.4.2 STANDBY mode typical wake-up timing

The following table shows the typical timing for STANDBY mode wake-up.

Table 21. Typical STANDBY mode wake-up timing

	HW wake-up sequence	CFLASH & DFLASH low-power exit	CFLASH & DFLASH RESET exit	FXOSC startup	FMPLL lock time	ADC power-Up
MPC56xx	787 µs	125 µs	125 µs	See Note[1]	40 µs	1.5 µs

1. For this STANDBY wake-up example, the HW wake-up sequence contains: ULPV stabilisation, FIRC startup, main VREG stabilisation and system initialization time.

The scope plot given in Figure 19 shows a device in STANDBY mode being wakened by a transition on a wake-up pin (blue line). An output is programmed to toggle at the first available line of code. This will not actually be the first line of code. Since the device was in STANDBY mode, all register content not in PD0 will have had power removed and thus, all register content will have been lost. Therefore, the first lines of code will be the initialization code which sets up pins, and modes.

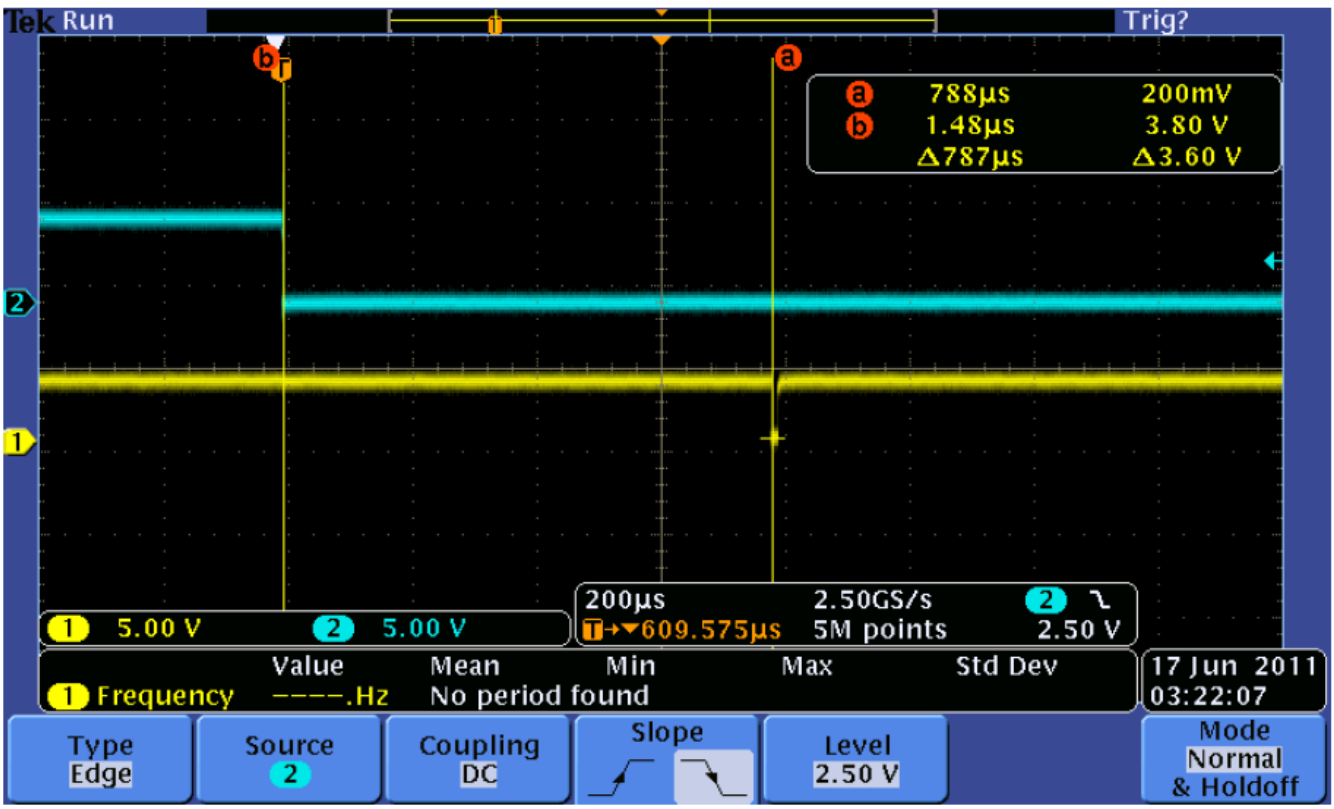


Figure 19. Exit from STANDBY

10.5.5 Wake-Up via RESET

The following figure shows the sequence of events after the internal voltage regulators have stabilized shown by the “power-on RESET” line.

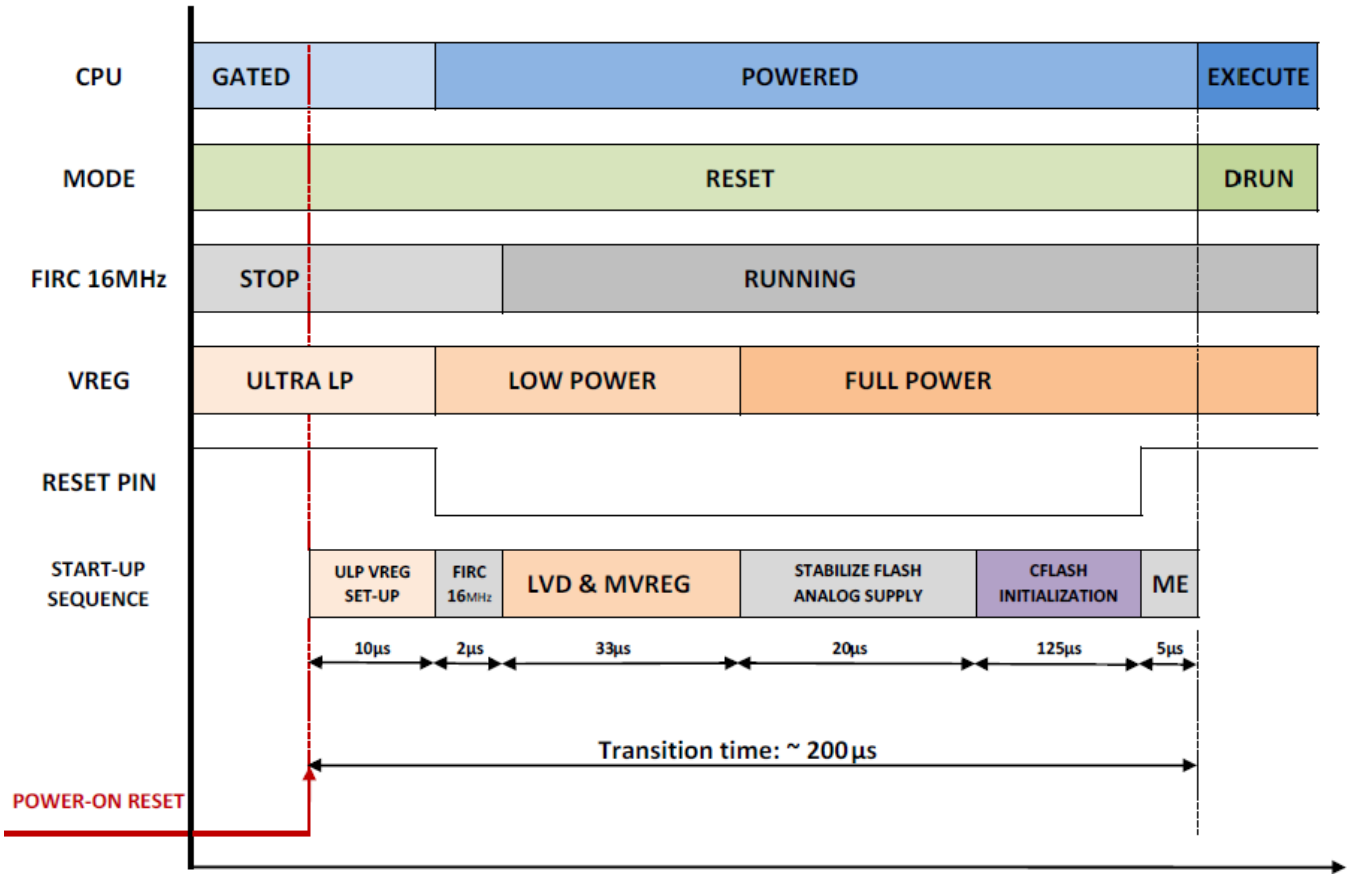
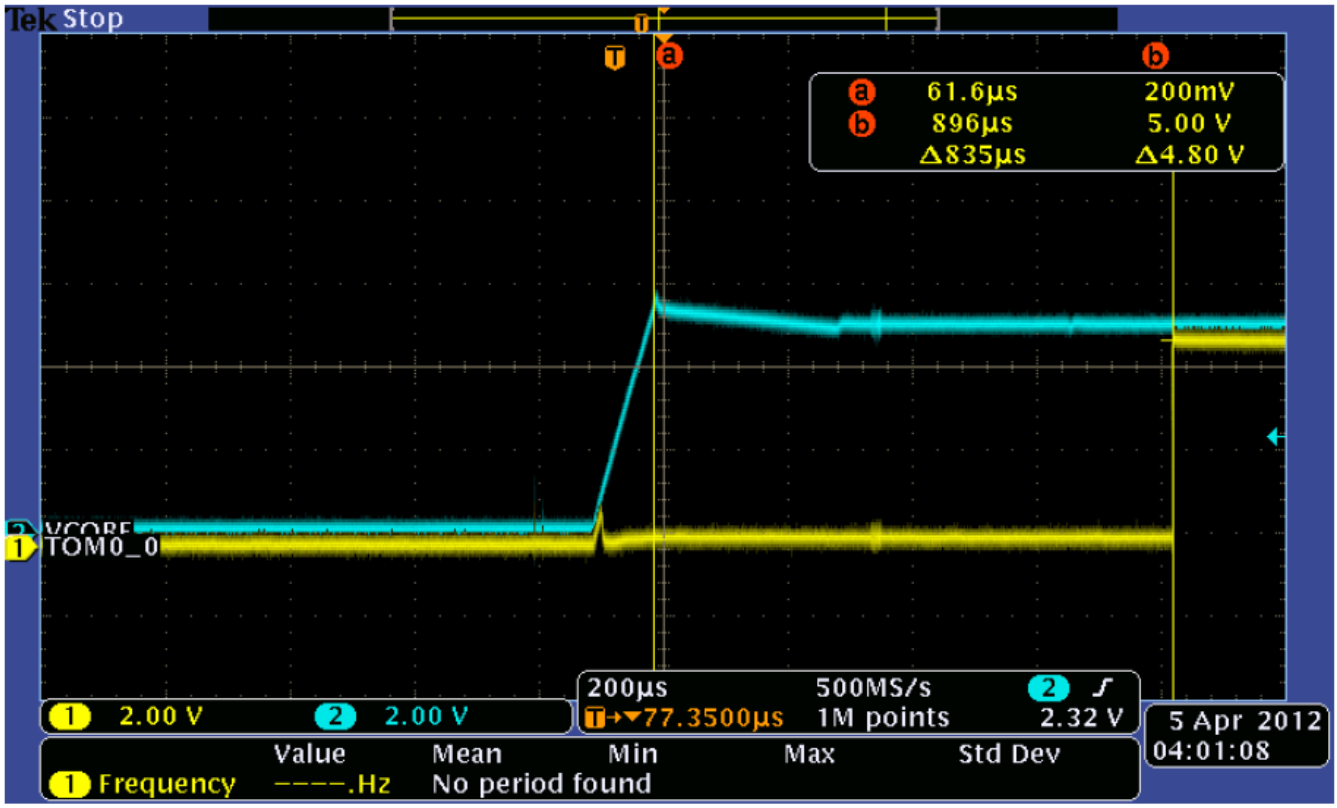


Figure 20. Typical MPC5604B/C power-on RESET timing

The scope plot below shows the VDD supply going high (blue line) and the RESET pin going high 835 µs later.



11 Current consumption

11.1 Measurements

From the MPC5604B Microcontroller Data Sheet, available at http://www.freescale.com, typical room temperature values for MPC5604B are as follows:

RUN at 64 MHz = 51mA

HALT (128 kHz IRC running) = 8 mA

STOP (128 kHz IRC running) = 180 µA

STANDBY (128 kHz IRC Running 32 KB RAM ON) = 30 µA

STANDBY (128 kHz IRC Running 8 KB RAM ON) = 20 µA

NOTE

See the data sheet for the particular family member being used, available at http://www.freescale.com for typical and maximum values. Always design systems based on the worst case maximum current.

12 Common mistakes

When evaluating new hardware and software, it is not unusual for the users to find themselves unable to achieve the low-power current in the Data Sheet of the particular device of MPC56xx family available at http://www.freescale.com and measured on Freescale evaluation boards. Sometimes current even increases at low-temperature instead of the proper behaviour of decreasing. The following checklist has been helpful for bringing up new hardware.

1. Ensure wakeup pins must have either internal or external pullups or pulldowns. If this is not the case, leakage will increase current. Even if one pin has no internal or external pullups or pulldowns, leakage current will increase. Make a list of all wake-up pins. Check the schematic to determine if an external pullup or pulldown exists. If not, enable weak pullup for that wake-up input in WKPU_WIPUER.
    a. Check WKPU_WIPUER bit positions. For example, WKUP0 in MPC5607B has register bit position 31, not 0.

NOTE

The only exception is that when a 32 kHz external oscillator is used, those pins should not have these internal pullups enabled.

2. Double check wake-up pins have internal or external pullups or pulldowns.
    a. Watch out for switched power to pullups/pulldowns. During low-power, wake-up inputs must be either below the input low-voltage threshold or above the high-input threshold.
    b. Examine effect of any voltage dividers to wake-up input voltage.
    c. Make sure resistors on the schematic are populated to wake-up inputs.
3. Ensure TDO pin has pullup or pulldown for STANDBY mode.
4. Check the device errata for low-power items
    a. For example, the document MPC5607B_1M03Y, Rev. 25 AUG 2011 errata e3242: PB[10], PD[0:1] pins configuration during STANDBY mode. Summary: These pins have both wake-up and analog functionality. The signal must be either than 0.2*VDD_HV or higher than 0.8*VDD_HV.

NOTE

Leaving the debugger connected during low-power will lead to increased current consumption.

13 Appendix: Debugging with low-power mode

By default, low-power modes shut off as much power as possible, including power to the debugger connection. To prevent the debugger from losing connection, and therefore, possibly asserting a reset, the Nexus Development Interface (NDI) module contains features to maintain a debugger connection. Debuggers may or may not implement this feature.

The controls to maintain the debugger connection are in the Port Configuration Register (PCR) of the Nexus Development Interface module. These registers are not memory-mapped, and are accessible only through the debugger. Appropriate debugger scripts must be used to configure these settings.

To maintain a debugger connection, the debugger must set the following:

When exiting STANDBY mode by pin transition or timer wake-up, there are two options for code execution depending on the configuration of RGM_STDBYBOOT_FROM_BKP_RAM:

- PCRLP_DBG_EN Low Power Debug Enable:

  Enables debug functionality to support exit from any low-power mode. Debug functionality is immediately available for breakpoints.

  When enabled, low-power exit takes longer because the state machine adds steps to wait until after a handshake to the debugger and the response of the debugger to the handshake is complete.

- PCRMCKO_EN MCKO Enable:

Enables clock which will be used in debugger connection.

- PCRMCKO_DIV MCKO Division Factor:

  Determines MCK0 relative to system clock frequency. Options are SYSCLK divided by 1 (default), 2, 4, or 8.

Table 22. Low-Power Sequence Summary

	Debugger	Software	Hardware
1	Enables debug functionality for low power mode exit by setting NDI_PCR[LP_DBG_EN]=1, as per user request		
2		The user requests low-power mode entry by writing to ME_MCTL	
			Device starts to enter low-power mode, and sets NDI_PCR[STOP_SYNC] = 1.[1]
3	Debugger senses the low-power mode entry by detecting NDI_PCR[STOP_SYNC] = 1		
4	Debugger saves context, and enables low-power entry to continue by writing NDI_PCR[STOP_SYNC] = 0		
5			Device enters low-power mode. SYSCLK is inactive. TDO is held high.
6			Wake-up event occurs, either by RTC/API or pin transition and sets TDO low.
7	Debugger senses TDO low, so it restores context and then sets STOP_SYNC]=1		
8			Device senses debugger handshake of NDI_PCR[STOP_SYNC] = 1, releases TDO being held low, and clears NDI_PCR[STOP_SYNC] = 0.
9		Software execution resumes	

1. STOP_SYNC is NDI_PCR bit 8. This bit is also known as LP1_SYN in some devices.

NOTE

- Leaving the debugger connected during low-power will lead to increased current consumption.
- Although STANDBY exit causes a reset, not all registers are reset as in a power-up reset. Power domain 0 registers are not reset, including: RGM registers, ME _DRUN_MC register and PCU registers, RTC registers, and CAN SAMPLER registers.
- The 32 kHz slow external crystal oscillator is always ON by default, but can be configured OFF in standby by setting the OSCON bit in the Slow External Crystal Oscillator Control Register.
  - Leaving the debugger connected during low power will lead to increased current consumption.

- After STANDBY entry, the debugger loses connection to the target due to power gating.
- After STANDBY exit, debugger resync takes longer as the MCU wake-up, that is, the breakpoints won't be recognized by the debugger, if placed close to the beginning of the STANDBY exit code.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number AN4240
Revision 1, 10/2017