NXP Semiconductors
Application Note

Document Number: AN12085
Rev. 1, 01/2019

How to use i.MX RT Low Power Feature

Contents

1. Introduction

This document describes the low-power application design points on i.MX RT series. Low power is an important feature on i.MX RT.

2. Chip overview

This section provides main features of i.MX RT chips.

2.1. i.MX RT chip overview

The i.MX RT is a Cortex-M7 based chip that operates at speeds up to 600 MHz to provide a high CPU performance and the best real-time response.

- Cortex-M7 based processor which can operates at speed up to 600 MHz.
- Advanced power management module with internal DCDC and LDO to reduce complexity of external power supply and simplifies power sequencing.
- Various memory interfaces, including SDRAM, Raw NAND FLASH, NOR flash, SD/eMMC.
- A wide range of other interfaces for connecting peripherals, such as WLAN, Bluetooth™, GPS, displays, and camera sensors.
- Rich audio and video features, including LCD display, basic 2D graphics, camera interface, S/PDIF and I2S audio interface.
- Provide rich peripheral modules, such as SPI, I2C, CAN, Ethernet, Flex-Timers, ADC, and so on.
- Target at Industrial HMI, Motor Control and Home Appliance areas.

3. Low-power overview

The following section describes the power supply, run mode, and low-power mode.

3.1. Power supply

Table 1 lists the external power supply rails of i.MX RT.

Table 1. External power supply rails

Power Rail	Description
DCDC_IN	Power for DCDC
VDD_HIGH_IN	Power for Analog
VDD_SNVS_IN	Power for SNVS and RTC
USB_OTG1_VBUS USB_OTG2_VBUS	Power for USB VBUS
VDDA_ADC	Power for 12-bit ADC
VDDA_IN	Power for LDO 2P5 and LDO 1P1
NVCC_SD0	Power for GPIO in SDIO1 bank (3.3 V mode)
	Power for GPIO in SDIO1 bank (1.8 V mode)
NVCC_SD1	Power for GPIO in SDIO2 bank (3.3 V mode)
	Power for GPIO in SDIO2 bank (1.8 V mode)
NVCC_GPIO	IO Power for GPIO in GPIO bank
NVCC_EMC	IO Power for GPIO in EMC bank

3.2. Run mode definition

Table 2 lists the run mode definitions.

Table 2. Run mode definition

Run Mode	Definition
Overdrive run	• CPU runs at 600 MHz, overdrive voltage to 1.275 V • Bus frequency at 150 MHz • All the peripheral are enabled and runs at target frequency
Full speed run	• CPU runs at 528 MHz, full loading, lower voltage to 1.15 V • Bus frequency at 132 MHz • All peripheral are enabled and runs at target frequency
Low speed run	• CPU runs at 132 MHz, lower voltage to 1.15 V • Bus frequency at 66 MHz • Some PLL are powered down • 20 % peripheral are active, others are in low power mode
Low power run	• CPU runs at 24 MHz, lower voltage to 0.95V • Bus frequency at 12 MHz • All PLL are powered down, XTAL 24 M powered down, OSC RC 24 M enabled • High-speed peripherals are power down

3.3. Run mode configuration

Table 3 describes the run mode configuration.

Table 3. Run mode configuration

	Overdrive Run	Full Speed Run	Low Speed Run	Low Power Run
CCM LPM Mode	RUN	RUN	RUN	RUN
CPU Core	600 MHz	528 MHz	132 MHz	24 MHz
L1 Cache	ON	ON	ON	ON
FlexRAM	ON	ON	ON	ON
SOC Voltage	1.275 V	1.15 V	1.15 V	0.95 V
Analog LDO	ON	ON	ON	In Weak Mode
24MHz XTAL OSC	ON	ON	ON	OFF
24MHz RC OSC	OFF	OFF	OFF	ON
System PLL	ON	ON	ON	OFF
Other PLLs	ON	ON	On as needed	On as needed
Module Clock	ON	ON	On as needed	Peripheral clock off
RTC32K	ON	ON	ON	ON

3.4. Low-power mode definition

Table 4 lists the low-power mode definition.

Table 4. Low-power mode definition

Low Power Mode	Definition
System Idle	• CPU can automatically enter this mode when no thread running • All the peripheral can remain active • CPU only enter WFI mode, it will have its state retained so the interrupt response can be very short
Low Power Idle	• Much lower power than System Idle mode, with longer exit time • All PLL are shut off, analog modules running in low power mode • All high-speed peripherals are power gated, low speed peripherals can remain running at low frequency
Suspend	• The most power saving mode with longest exit time • All PLLs are shut off, XTAL are off, all clocks are shut off except 32 K clock • All high-speed peripherals are power gated, low speed peripherals are clock gated
SNVS	• All SOC digital logic, analog modules are shut off only except SNVS domain • 32 KHz RTC is alive

3.5. Low power mode configuration

Table 5 describes the low-power mode configuration.

Table 5. Low-power mode configuration

	System Idle	Low Power Idle	Suspend	SNVS
CCM LPM Mode	WAIT	WAIT	STOP	-
Arm Core (PDM7)	WFI	WFI	Power Down	OFF
L1 Cache	ON	ON	Power Down	OFF
FlexRAM (PDRET)	ON	ON	ON	OFF
FlexRAM (PDRAM0)	ON	ON	Power Down	OFF
FlexRAM (PDRAM1)	ON/OFF	ON/OFF	Power Down	OFF
VDD_SOC_IN Voltage	1.15 V	0.95 V	0.925 V	OFF
SYS PLL	ON	Power Down	Power Down	OFF
Other PLL	Power Down	Power Down	Power Down	OFF
24MHz XTAL OSC	ON	OFF	OFF	OFF
24MHz RC OSC	OFF	ON	OFF	OFF
LDO 2P5	ON	OFF	OFF	OFF
LDO 1P1	ON	OFF	OFF	OFF
WEAK 2P5	OFF	ON	OFF	OFF
WEAK 1P1	OFF	ON	OFF	OFF
Band gap	ON	OFF	OFF	OFF
Low-Power Band gap	ON	ON	ON	OFF
AHB Clock	33 MHz	12 MHz	OFF	OFF
IPG Clock	33 MHz	12 MHz	OFF	OFF
PER Clock	33 MHz	12 MHz	OFF	OFF
Module Clocks	ON as needed	ON as needed	OFF	OFF
RTC 32K	ON	ON	ON	ON

NOTE

Before entering low power mode, each module is enabled or disabled as described in the configuration table. However, for wake-up ensure to restore the settings.

3.5.1. Wake-up source

Table 5 describes the wake-up source.

Table 6. Wake-up source

	System Idle	Low Power Idle	Suspend	SNVS
GPIO wake-up	YES	YES	YES	- YES (1 PIN only)
RTC wake-up	YES	YES	YES	YES
USB remote wake-up	YES	YES	YES	NO
Others wake-up source	YES	YES	ON	NO

NOTE

No matter whether the mode is System Idle, Low Power Idle, or Suspend, user has to enable the wake-up interrupt in GPC module else the wake-up fails.

The only pin that can wake-up the system in SNVS is GPIO5_IO00.

USB remote wake-up is not described in this document.

4. Power mode switch example

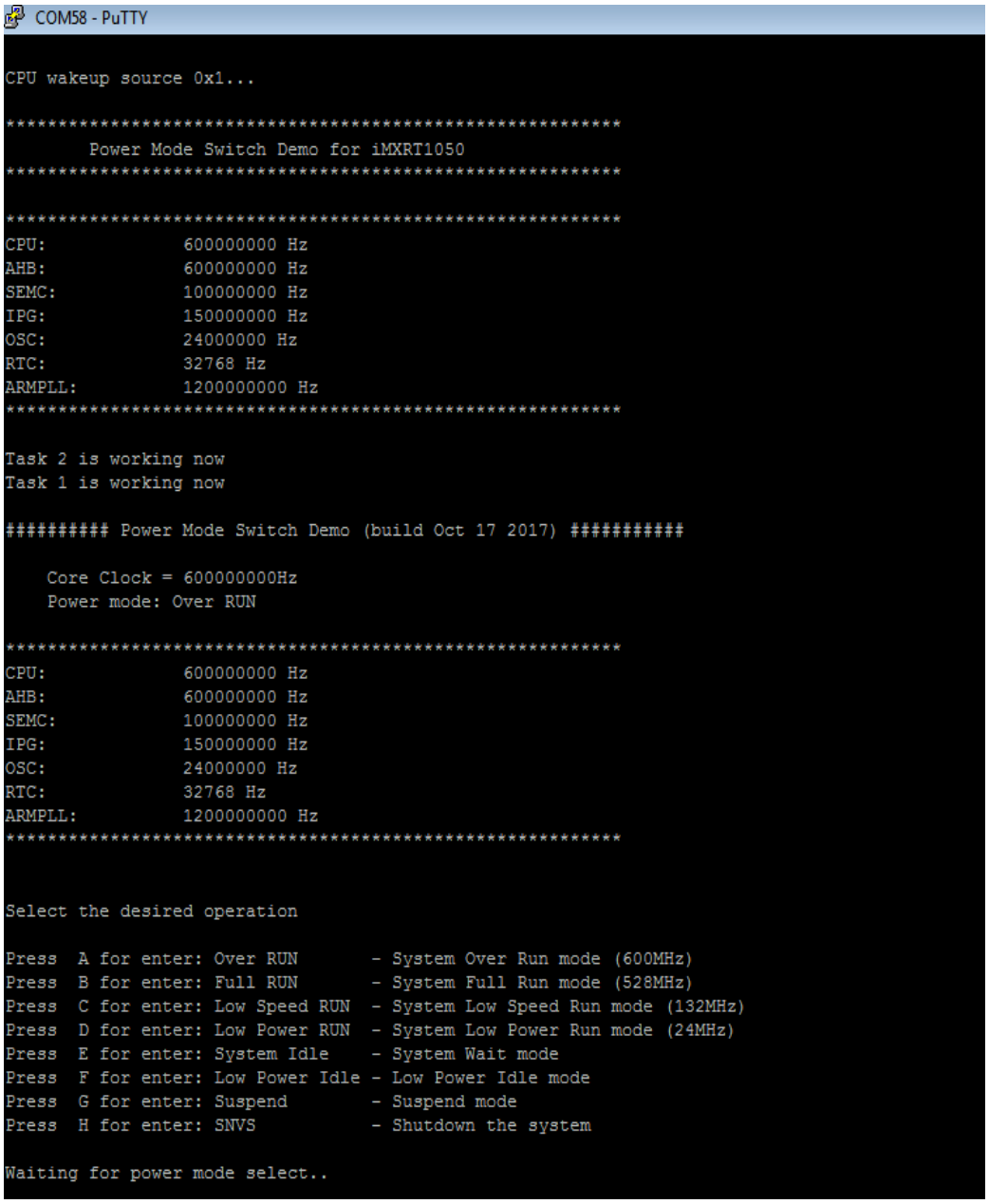
This section provides the application description, power mode switch menu

The power mode switch example is designed to simulate a customer low power application case.

Features:

- FreeRTOS and tickless feature support.
- Support all RUN and low power modes switch.
- Support run after WFI instruction and reset in suspend reset (controlled by chip).
- Simulate two running tasks and show task status in power mode switch.

4.1.1. Power mode switch menu



In this menu, user can switch power modes and wake-up from idle, suspend modes.

4.1.2. Task status show

When user select to transfer to a new power mode. The task status is shown.

1. Start working

```
Task 2 is working now
Task 1 is working now
```

2. Power mode transition
   For example,
   Overrun => System Idle

```
Waiting for power mode select..

WorkingTask 2: Transfer from Over RUN to System Idle
WorkingTask 1: Transfer from Over RUN to System Idle
```

3. Wake-up transfer back to Overrun

```
Waiting for key press..

Switch SW8 from off to on to wake up.
WorkingTask 2: Transfer from System Idle to Over RUN
WorkingTask 1: Transfer from System Idle to Over RUN

Next loop
```

4.1.3. Wake-up source

Two wake-up sources can be selected for System Idle, Low Power Idle and Suspend mode, SW8 key (on EVK board) and GPT timer.

```
Waiting for power mode select..

WorkingTask 2: Transfer from Over RUN to System Idle
WorkingTask 1: Transfer from Over RUN to System Idle
Select the wake up source:
Press T for GPT - GPT Timer
Press S for switch/button SW8.
```

- SW8 user key wake-up.
- GPT timer wake-up.

When choose GPT timer as wake-up source, another menu will ask user to input a duration from 1s to 9s to wake up.

.

```
Select the wake up timeout in seconds.
The allowed range is 1s ~ 9s.
Eg. enter 5 to wake up in 5 seconds.

Waiting for input timeout value...

3
Will wakeup in 3 seconds.
WorkingTask 2: Transfer from Low Power Idle to Over RUN
WorkingTask 1: Transfer from Low Power Idle to Over RUN

Next loop
```

5. Low power application design

This chapter will give some guides in designing a low power application. Also, we'll give some code examples about low power module enable and disable.

5.1. General description

Enter a low power mode is a set of enable/disable modules operations. The low power mode configuration should be paid attention.

Table 7. Low power configuration

	System Idle	Low Power Idle	Suspend	SNVS
CCM LPM Mode	WAIT	WAIT	STOP	-
Arm Core (PDM7)	WFI	WFI	Power Down	OFF
L1 Cache	ON	ON	Power Down	OFF
FlexRAM (PDRET)	ON	ON	ON	OFF
FlexRAM (PDRAM0)	ON	ON	Power Down	OFF
FlexRAM (PDRAM1)	ON/OFF	ON/OFF	Power Down	OFF
VDD_SOC_IN Voltage	1.15 V	0.95 V	0.925 V	OFF
SYS PLL	ON	Power Down	Power Down	OFF
Other PLL	Power Down	Power Down	Power Down	OFF
24MHz XTAL OSC	ON	OFF	OFF	OFF
24MHz RC OSC	OFF	ON	OFF	OFF
LDO 2P5	ON	OFF	OFF	OFF
LDO 1P1	ON	OFF	OFF	OFF
WEAK 2P5	OFF	ON	OFF	OFF
WEAK 1P1	OFF	ON	OFF	OFF
Band gap	ON	OFF	OFF	OFF
Low Power Band gap	ON	ON	ON	OFF
AHB Clock	33 MHz	12 MHz	OFF	OFF
IPG Clock	33 MHz	12 MHz	OFF	OFF
PER Clock	33 MHz	12 MHz	OFF	OFF
Module Clocks	ON as needed	ON as needed	OFF	OFF
RTC32K	ON	ON	ON	ON

5.2. Low power mode enter and exit sequence

On i.MX RT, the chip can enter each low power mode and exit to rum mode.

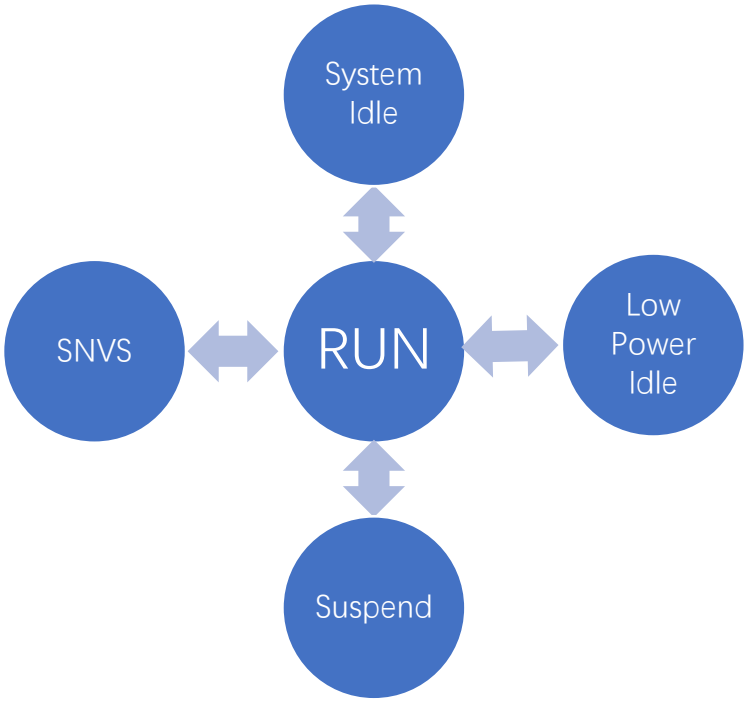


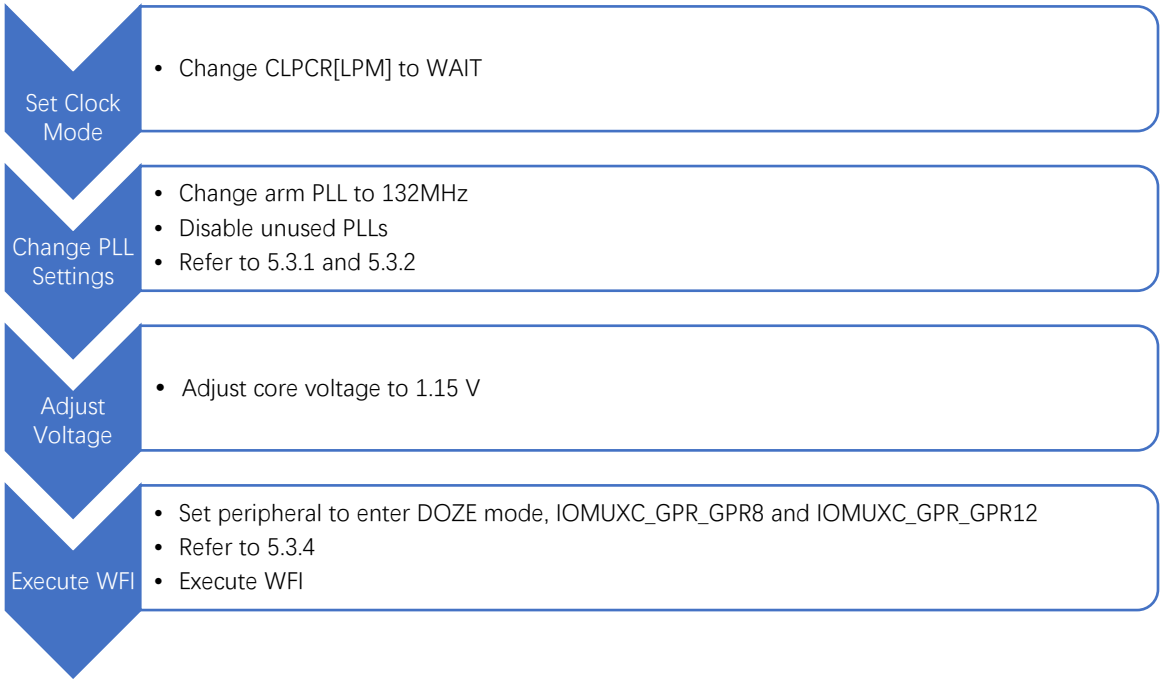
Figure 1. Low Power Mode Enter and Exit Sequence

5.2.1. Enter system idle mode

System Idle mode is a simple mode that the most modules' default settings do not need to be changed.

5.2.1.1. How to enter

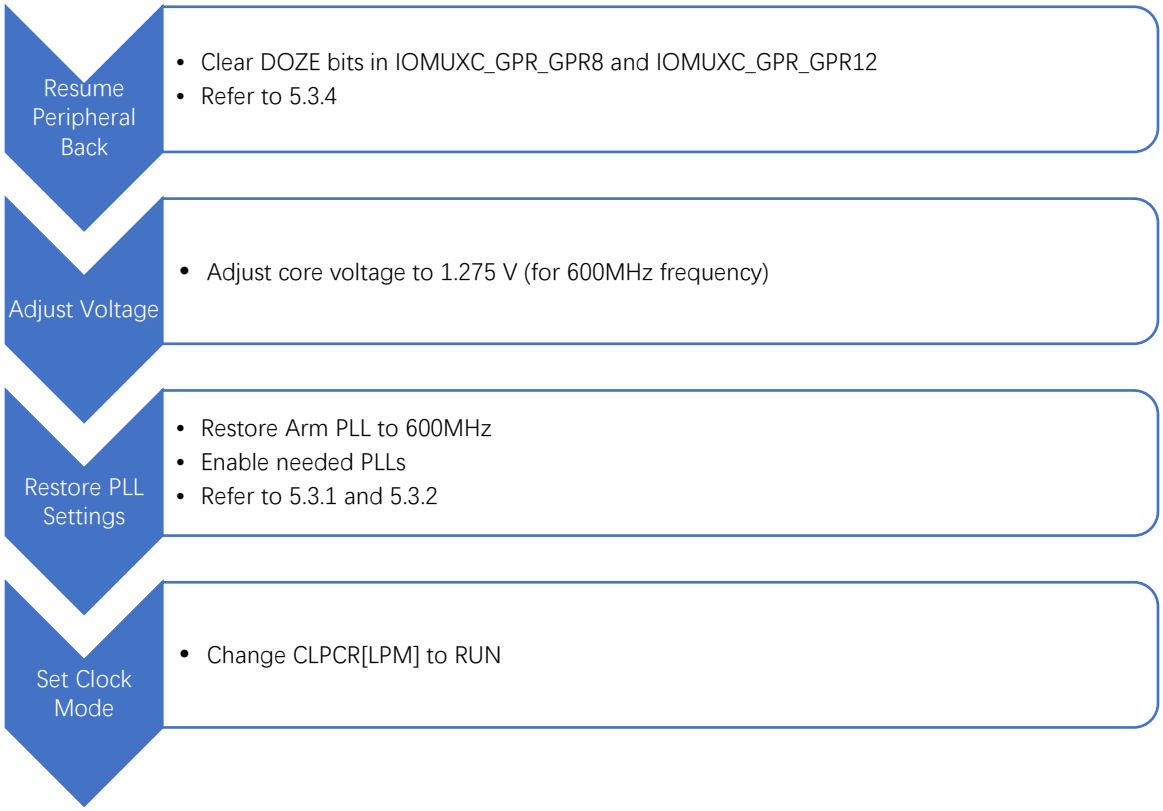
To enter system idle mode, the core frequency should be changed first and then unused PLLs should be powered down. Finally, the Core voltage also needs to be changed accordingly. Before executing WFI instruction, the used modules' DOZE bits need to be set.



Assume that the core voltage is reduced to 132MHz. If the core frequency switches from 24MHz to 132MHz, the core voltage should be increased first.

5.2.1.2. How to exit

To exit of System Idle mode is also simple. Adjust the core voltage first then enable PLLs.



5.2.2. Enter low power idle mode

5.2.2.1. How to enter

To enter Low Power Idle mode refer to the diagram below.


Set Clock Mode
• Change CLPCR[LPM] to WAIT
Switch DC-to-DC
• Switch DCDC to use internal OSC
Power Down USBPHY
• Power down USBPHY
Change PLL Settings
• Change Arm PLL to use 24 MHz OSC
• Disable all other PLLs
• Refer to 5.3.1 and 5.3.2
Enable Weak LDO
• Enable weak 2P5 and 1P1 LDO
• Turn off regular 2P5 and 1P1 LDO
• Refer to 5.3.5
Switch to RC-OSC
• Switch to 24 MHz RC OSC
• Turn off external 24 Mhz OSC
Adjust Voltage
• Adjust core voltage to 0.95 V
Disable Regular Band gap
• Disable regular band gap and enable low power band gap
• Refer to 5.3.7
Execute WFI Instruction
• Set peripheral to enter DOZE mode, refer to 5.3.4
• Execute WFI

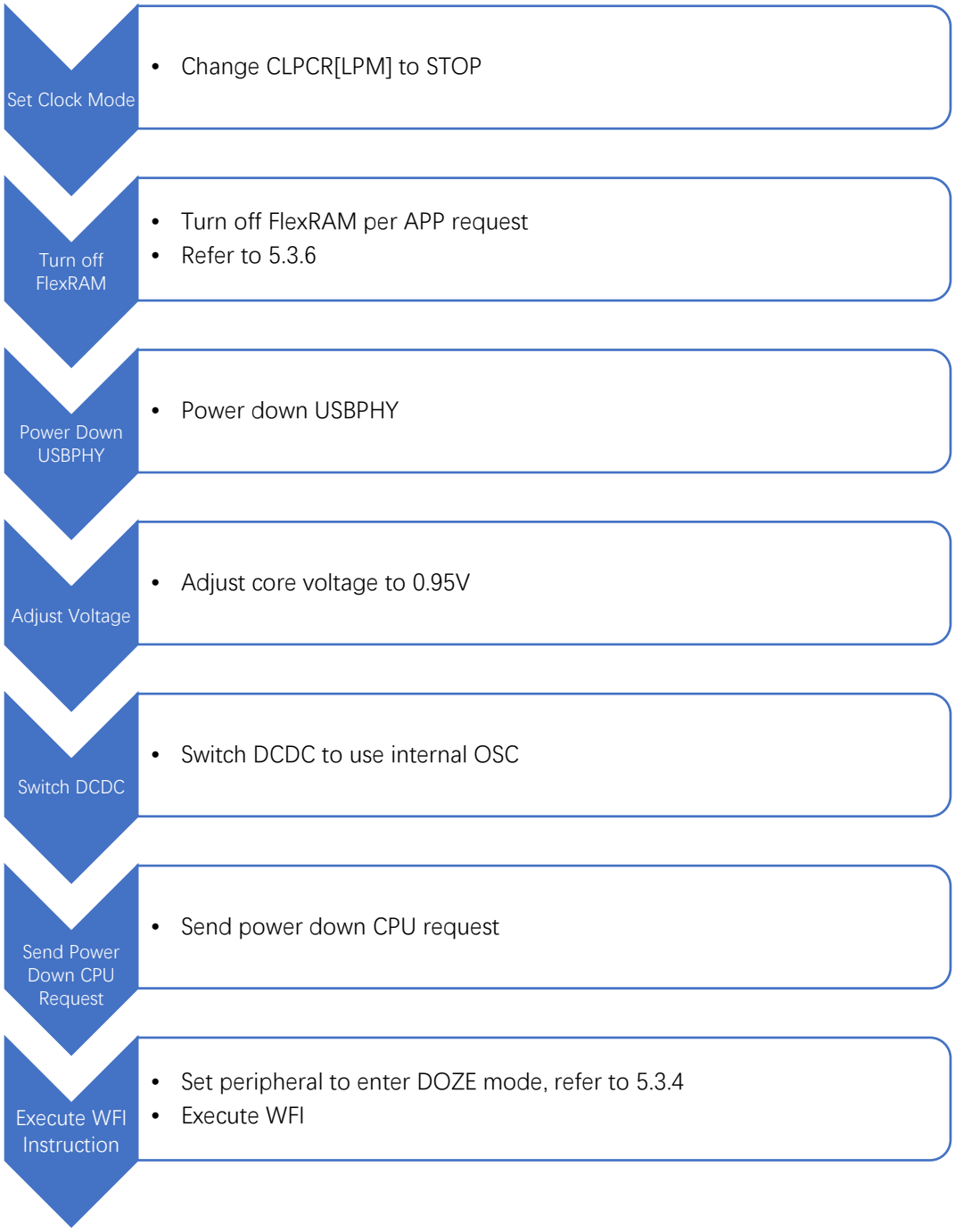
5.2.2.2. How to exit

To exit Low Power Idle mode, refer to the diagram below.

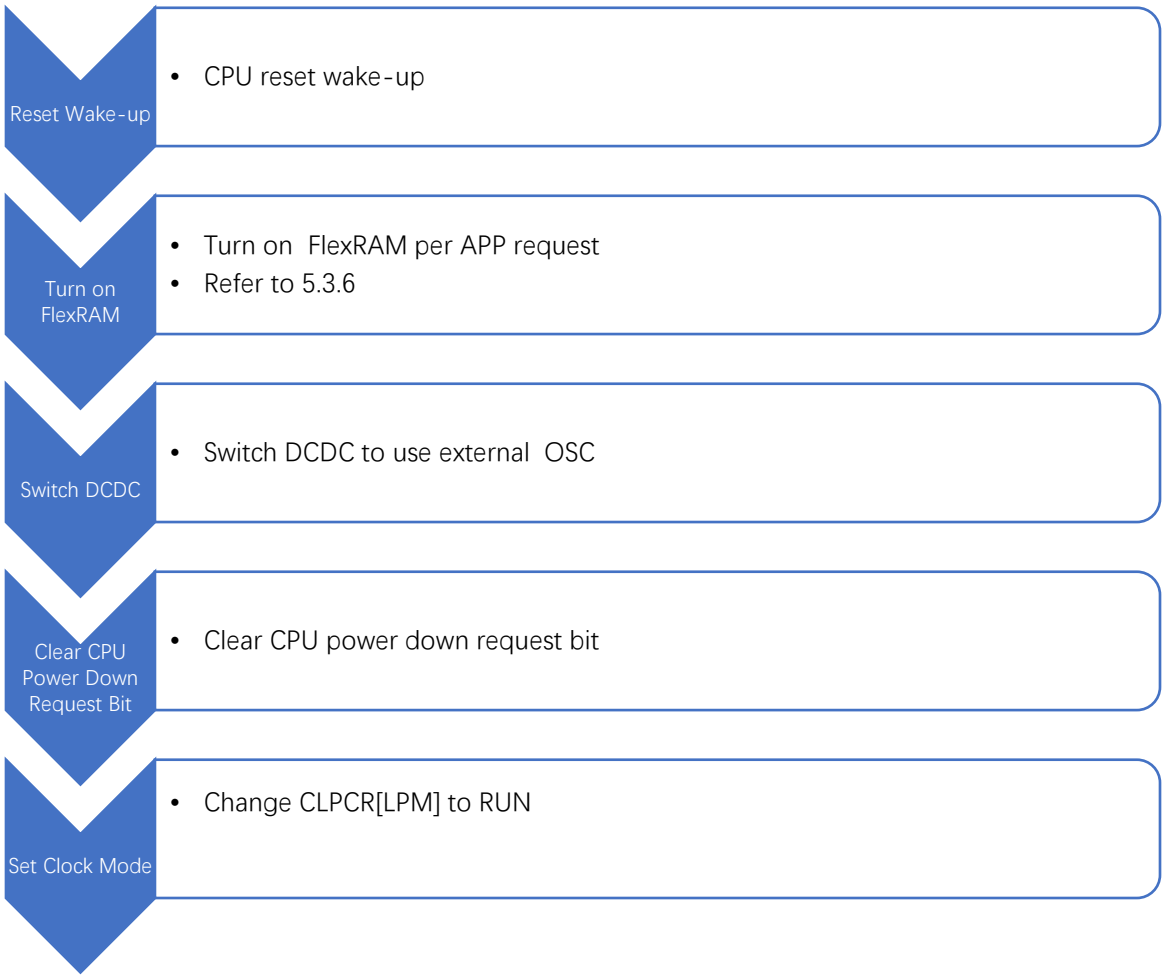
5.2.3. Enter suspend mode

In suspend mode, as when set CLPCR[LPM] to STOP, the system will power down PLL don't need to adjust frequency.

5.2.3.1. How to enter



5.2.3.2. How to exit



5.2.4. SNVS mode

5.2.4.1. How to enter

There are two methods to enter SNVS Mode. Software and Hardware.

Software to enter SNVS mode, set SNVS_LPCR[TOP] to 1.

Example code:

```
SNVS->LPCR |= SNVS_LPCR_TOP_MASK;
while (1) /* Shutdown */
    {
    }
```

Hardware to enter SNVS mode, long pressing ON/OFF button (SW2 on EVK Board)

5.2.4.2. How to exit

Long pressing ON/OFF button (SW2 on EVK board) or using the wake-up pin.

This chip supports the use of a WAKEUP pin (GPIO5_IO00 ALT5) to request main SoC power on to exit SNVS mode. To use it, follow the tips below:

- SNVS must be in Dumb PMIC mode (default, and must for on-chip DCDC)
- Configure IOMUXC_SNVS to select WAKEUP pin ALT5 to mux it to GPIO5_IO00
- Configure GPIO5 ICR to either low level or high level sensitive
- Set GPIO5 IMR bit0 to enable GPIO5_IO00 interrupt
- Enter SNVS mode by SW or ON/OFF button, then on-chip DCDC will be off, and PMIC_ON_REQ pin will also go to low to notify outside
- Assert WAKEUP pin for at least two 32 kHz cycles (active high or low depends on GPIO5 ICR configuration), so that GPIO5 interrupt gets sampled by SNVS module
- SNVS will assert PMIC_ON_REQ pin high and on-chip DCDC begins to ramp on
- SoC power on procedure (except SNVS) begins (ROM boot, …)

5.3. Low power design point

5.3.1. How to change core frequency

When the frequency of Core needs to change the voltage of the Core also need to be change. There is a rule in changing core frequency and voltage.

- Increasing core frequency, voltage first, then frequency.
- Decreasing core frequency, frequency first, then voltage.

5.3.2. How to change PLL frequency

The sequence in changing PLL frequency is listed below:

- Set BYPASS and ENABLE bits to bypass clock.
- Set PLL clock divider.
- Clear PLL BYPASS bit.
- Wait for CCM operation finishes by testing CCM_CDHIPR register

Example code:

```
/* Wait CCM operation finishes */
        while (CCM->CDHIPR != 0)
        {
        }
```

NOTE

When the frequency of a PLL needs to be changed, BYPASS clock first, then change PLL frequency and at last switch PLL back.

5.3.3. How to change clock source for SPI Flash and SDRAM

Usually under any RUN mode, if the SPI Flash or SDRAM are used to store code or data, changing the root clock's frequency is not recommended. It is dangerous to R/W SPI Flash or SDRAM while changing their clock source and frequency. If these operations have to do, the code should be run in On-Chip RAM.

5.3.4. How to enable tickless low power feature in FreeRTOS

The FreeRTOS tickless idle mode stops the periodic tick interrupt during idle periods (periods when there are no application tasks that are able to execute), then makes a correcting adjustment to the RTOS tick count value when the tick interrupt is restarted.

Stopping the tick interrupt allows the microcontroller to remain in a deep power saving state until an interrupt occurs.

NOTE

MCU-SDK have already provided fsl_tickless_systick.c and fsl_tickless_gpt.c to implement tickles low power feature. User need to define configUSE_TICKLESS_IDLE in FreeRTOSConfig.h.

5.3.4.1. Choose a FreeRTOS tick timer

Normally, the system tick module is used as the tick timer in FreeRTOS. On i.MX RT series the system tick module in Cortex-M7 core can support two timer sources, core clock and 100KHz clock, which is derived from 24M oscillator and divided to 100KHz.

As in FreeRTOS, the time consumed in tickless stop will be compensated. The time consumed in tickles stop should not exceed maximum time duration (or FreeRTOS won't be able to know how much time to compensate). The code to calculate maximum duration counts is in function vPortSetupTimerInterrupt() in tickless code.

```
ulTimerCountsForOneTick = ( configSYSTICK_CLOCK_HZ / configTICK_RATE_HZ );
```

```
xMaximumPossibleSuppressedTicks = portMAX_32_BIT_NUMBER / ulTimerCountsForOneTick;
```

For system tick module, *portMAX_32_BIT_NUMBER* is 0x00FFFFFF and normally *configTICK_RATE_HZ* is set to 1000, meaning that each time tick duration is 1ms.

So, in case choose core clock 600MHz as clock source for system tick module, *ulTimerCountsForOneTick* will be 600000 and *xMaximumPossibleSuppressedTicks* will be 27.962025, which is also 27.962025 ms.

27.962025 ms is too short for a tickless duration. It means when in tickles stop, the system need to wake up every 27.962025 ms to compensate the time.

So, when consider using tickles low power in a FreeRTOS application, we'd better not use core clock as systick clock source, but use 100KHz clock source. In this case, *ulTimerCountsForOneTick* will be 100 and

xMaximumPossibleSuppressedTicks will be 167772.15 ms, which is about 168s. It's an acceptable duration here.

To use 100KHz clock just need to define configSYSTICK_CLOCK_HZ in FreeRTOSConfig.h.

#define configSYSTICK_CLOCK_HZ (10000U)

5.3.4.2. Choose a tickless timer

As in tickless mode, FreeRTOS also need to compensate the consumed time, we need to use a timer with maximum time ticks that FreeRTOS can hold.

Systick is the first choice for this timer. But in i.MX RT, systick interrupt is not imported in GPC module. So systick interrupt can't wake-up the system, but only the Arm core. We need to select another timer as tickless timer.

The requirement for a tickless timer will be:

- Timer.
- Imported in GPC and can wake up the system.
- Work in System Idle, Low Power Idle and Suspend mode, even when core and all PLLs are power down.

GPT module can use 32K RTC (by setting GPT_CR[CLKSRC] = 4) as clock source which can work in System Idle, Low Power Idle and Suspend mode.

5.3.4.3. Hook functions before and after WFI instruction

FreeRTOS tickless idle mode provide two methods.

- vPortSuppressTicksAndSleep function.
  In tickless mode, the system will call vPortSuppressTicksAndSleep function in idle task and pass the expect sleep time in microsecond as parameter.

User can implement a customized vPortSuppressTicksAndSleep and Idle task will call this function periodically.

- Advantage: Can handle WFI instruction as user required.
  For example, as in suspend mode, the core is power down and thus, the wake-up is indeed a core reset. But as context can be saved, user can run code from WFI after reset. User need to setup a restore point to restore when reset.
  In this case, it is easier for user to handle the WFI instruction in customer function.
- Disadvantage: User function need to calculate the tickles period and call vTaskStepTick to compensate the consumed time.
  Note: User can take the vPortSuppressTicksAndSleep function in fsl_tickless_systick.c and fsl_tickless_gpt.c as a reference to compensate the time.
- Two hook functions to call before and after WFI instruction, configPRE_SLEEP_PROCESSING and configPOST_SLEEP_PROCESSING.

If user want to reuse vPortSuppressTicksAndSleep in fsl_tickless_systick.c and fsl_tickless_gpt.c, configPRE_SLEEP_PROCESSING and configPOST_SLEEP_PROCESSING can be implemented to change and restore modules for a lower power consumption.

- Advantage: Don't need to care about time compensation. vPortSuppressTicksAndSleep function in fsl_tickless_systick.c and fsl_tickless_gpt.c will do it for you.
- Disadvantage: Not very flexible. User can't proceed WFI instruction in own function.

In Power Mode Switch example, we use the second method to reuse vPortSuppressTicksAndSleep.

5.3.5. Doze and IPG STOP signal

In order to reduce power consumption the Doze and Stop bits should be set and ask each module to enter Doze of Stop status before entering low power mode. In the example code, all supported peripherals' DOZE bits are enabled, but in an application the developer should know what modules are needed and enable corresponding modules' DOZE bits.

5.3.5.1. Doze mode

For each module in doze mode, the AHB clock and serial clock domain will be gated off internally, but IPS Bus clock is not gated off. This mode is entered by setting corresponding DOZE bits in IOMUXC_GPR_GPR8 and IOMUXC_GPR_GPR12 and exited by clear corresponding bits in IOMUXC_GPR_GPR8 and IOMUXC_GPR_GPR12.

NOTE

The Doze mode works in all low power modes. So the Doze bits should be set before entering any low power mode.

Example code:

Before entering low power mode:

```
IOMUXC_GPR->GPR8 = 0xaaaaaaaa;
        IOMUXC_GPR->GPR12 = 0x0000000a;
```

After wake-up:

```
IOMUXC_GPR->GPR8 = 0x00000000;
        IOMUXC_GPR->GPR12 = 0x00000000;
```

5.3.5.2. Stop mode

When system requires a peripheral device to enter stop mode, it will wait for all transactions to complete and return ACK handshake to system. After ACK handshake returned, the system can gate the AHB Bus clock, IPG Bus clock and serial clock in system level. There is no clock gating action internally. The mode is entered by setting corresponding **DOZE and STOP bits** in **IOMUXC_GPR_GPR8** and IOMUXC_GPR_GPR12, and system should wait for the accordingly bits in **IOMUXC_GPR_GPR4 and IOMUXC_GPR_GPR7** to be set. The Suspend mode works under stop mode. So that the Stop and Doze bits should be set before entering Suspend mode.

In the example code, all supported peripherals' STOP and DOZE bits are enabled, but in an application, the developer should be aware of what modules are needed and enable corresponding modules' STOP and DOZE bits.

Example code:

Before entering suspend mode:

```
IOMUXC_GPR->GPR4 = 0x00000011;
while ((IOMUXC_GPR->GPR4 & 0x00110000) != 0x00110000){};
IOMUXC_GPR->GPR4 = 0x000036ff;
IOMUXC_GPR->GPR7 = 0x0000ffff;
IOMUXC_GPR->GPR8 = 0xfffcffff;
IOMUXC_GPR->GPR12 = 0x0000000a;
while ((IOMUXC_GPR->GPR4 & 0x36f90000) != 0x36f90000){};
while ((IOMUXC_GPR->GPR7 & 0xffff0000) != 0xffff0000){};
```

After wake-up:

```
IOMUXC_GPR->GPR4 = 0x00000000;
IOMUXC_GPR->GPR7 = 0x00000000;
IOMUXC_GPR->GPR8 = 0x00000000;
IOMUXC_GPR->GPR12 = 0x00000000;
```

Looking at the codes above, the STOP request is sent to EDMA and ENET first.

```
IOMUXC_GPR->GPR4 = 0x00000011;
while ((IOMUXC_GPR->GPR4 & 0x00110000) != 0x00110000){};
```

The reason is that EDMA and ENET are two modules that can act as a master on BUS and initiate transfer. If the code run on SPI Flash, the stop request shouldn't send to SPI Flash.

NOTE

Ensure that before entering Suspend mode, if corresponding module's STOP request is asserted, the system should wait for the acknowledge bit to be set. Or the system may fail to enter the Suspend mode.

If the code run in SDRAM and user wants to enter Suspend mode. SDRAM needs to wait a STOP signal to enter self-refresh mode. Here if a STOP request is sent to SDRAM and polling for the acknowledge bit to be asserted, the bit asserted might never get . Because the code is running on SDRAM and the transaction on SDRAM can't STOP. In this case, the code or data should be run in SPI Flash or On-chip RAM.

NOTE

Depending on whether clock gate and IP module's enable bit is set, each IP module may have a different behavior when the stop signal send to it.

- Some modules can acknowledge to stop signal even clock gate is OFF, for example PIT.
- Some modules can't acknowledge to stop signal when clock gate is OFF. But when clock gate is ON, it can acknowledge to stop signal, for example EDMA.
- Some modules can acknowledge to stop signal when clock gate is ON and the module's enable bit is ON. Currently only CAN is of this type.

5.3.6. How to enable weak analog LDO

There are two types Analog LDO on i.MX RT, regular LDO and low power weak LDO. The 1.1 V and 2.5 V LDO includes an alternate, self-biased, low-precision, weak regulator which can be enabled for applications needing to keep the 1.1 V and 2.5 V output voltage alive during low-power modes where the main regulator and its associated global band gap reference module are disabled. Using the weak LDO can reduce power consumption under Low Power Idle mode.

- Enable Weak Analog LDO

```
/* Enable regular 2P5 and wait it stable */
PMU->REG_2P5_SET = PMU_REG_2P5_ENABLE_LINREG_MASK;
while ((PMU->REG_2P5 & PMU_REG_2P5_OK_VDD2P5_MASK) == 0)
 {
 }
/* Turn off weak 2P5 */
PMU->REG_2P5_CLR = PMU_REG_2P5_ENABLE_WEAK_LINREG_MASK;

/* Enable regular 1P1 and wait for stable */
PMU->REG_1P1_SET = PMU_REG_1P1_ENABLE_LINREG_MASK;
while ((PMU->REG_1P1 & PMU_REG_1P1_OK_VDD1P1_MASK) == 0)
{
```

```
}
/* Turn off weak 1P1 */
PMU->REG_1P1_CLR = PMU_REG_1P1_ENABLE_WEAK_LINREG_MASK;
```

- Disable Weak Analog LDO

```
/* Enable weak 2P5 and turn off regular 2P5 */
PMU->REG_2P5 |= PMU_REG_2P5_ENABLE_WEAK_LINREG_MASK;
PMU->REG_2P5 &= ~PMU_REG_2P5_ENABLE_LINREG_MASK;
/* Enable weak 1P1 and turn off regular 1P1 */
PMU->REG_1P1 |= PMU_REG_1P1_ENABLE_WEAK_LINREG_MASK;
PMU->REG_1P1 &= ~PMU_REG_1P1_ENABLE_LINREG_MASK;
```

5.3.7. How to disable OCRAM power

The i.MX RT1050 chips have On-Chip FlexRAM which is shared by I-TCM, D-TCM and general purpose On-Chip RAM (OCRAM). The FlexRAM manages a big on-chip RAM array. The FlexRAM have 16 RAM banks, which are divided into Bank0, FlexRAM0 (Bank 1 - 7) and FlexRAM1 (Bank 8 -15). Description of OCRAM banks is listed below.

Table 8. Description of OCRAM banks

	Bank 0	**Bank 1-7**	**Bank 8-15**
Alias Name	Bank0	FlexRAM0	FlexRAM1
Power Domain	SNVS	PDRAM0	PDRAM1
Power Gate Bit	N/A	GPC_CNTR[PDRAM0_PGE]	PGC_MEGA_CTRL[PCR]
Power Gate Default Value	N/A	1 (Will power down in low power mode)	0 (Won’t power down)
LPM MODE ON/OFF Status	ON	Controllable	Controllable
Power down moment	In SNVS mode, bank0 is power down.	GPC_CNTR[PDRAM0_PGE] is set and Bank 1-7 will be off when core executes WFI instruction	PGC_MEGA_CTRL[PCR] is set and Bank 8-15 will be off immediately

NOTE

Pay attention to that when the power gate bit is set, FlexRAM0 will not be OFF immediately, unless core execute WFI instruction. But FlexRAM1 will be OFF immediately.

Code examples:

- Power Down FlexRAM0 and FlexRAM1

```
/* Turn off FlexRAM1 */
PGC->MEGA_CTRL |= PGC_MEGA_CTRL_PCR_MASK;
/* Turn off FlexRAM0*/
GPC->CNTR |= GPC_CNTR_PDRAM0_PGE_MASK;
```

- Power on FlexRAM0 and FlexRAM1

```
/* Turn on FlexRAM1 */
PGC->MEGA_CTRL &= ~PGC_MEGA_CTRL_PCR_MASK;
/* Turn on FlexRAM0*/
GPC->CNTR &= ~GPC_CNTR_PDRAM0_PGE_MASK;
```

5.3.8. Band gap switch

In i.MX RT, there are two band gap modules, regular band gap and low power band gap. In a low-power application, the low power band gap should be used before entering low power wait or stop state.

NOTE

In suspend mode, STOP mode will be set in CLPCR, which will help to switch band gap to low power band gap automatically in low power stop.

Code example:

Switch band gap to low power band gap:

```
/* Switch band gap */
  PMU->MISC0_SET   = 0x00000004;
  XTALOSC24M->LOWPWR_CTRL_SET = XTALOSC24M_LOWPWR_CTRL_LPBG_SEL_MASK;
PMU->MISC0_SET   = CCM_ANALOG_MISC0_REFTOP_PWD_MASK;
Switch band gap back to regular band gap:
/* Restore band gap */
    /* Turn on regular band gap and wait for stable */
    CCM_ANALOG->MISC0_CLR = CCM_ANALOG_MISC0_REFTOP_PWD_MASK;
    while ((CCM_ANALOG->MISC0 & CCM_ANALOG_MISC0_REFTOP_VBGUP_MASK) == 0)
    {
    }
    /* Low power band gap disable */
    XTALOSC24M->LOWPWR_CTRL_CLR = XTALOSC24M_LOWPWR_CTRL_LPBG_SEL_MASK;
    PMU->MISC0_CLR   = 0x00000004;
```

5.3.9. Enable DCM mode of DCDC

DCDC module provides a Discontinuous Conduction Mode (DCM) and Continuous Conduction Mode (CCM). It can operate on the CCM mode or DCM mode under Run mode. Which mode is used depends on the level of the load current. Usually the CCM mode is used when the chip under Run mode and DCM mode is used when the chip under low power mode. The DCM mode also can be used if the power consumption is low under RUN mode. DCM mode can reduce current of DCDC_IN, but DCDC_OUT current will not change.

NOTE

Because discontinuous conduction mode(DCM) can increase the efficiency of DCDC in case of low current loading, it is always recommended.

The DCM mode should be set before entering low power mode. The details steps as flowing:

```
pwd_zcd=0x0
pwd_cmp_offset=0x0
loopctrl_en_rcscale=0x4
DCM_SET_CTRL=1'b1
```

Don’t forget to restore to CCM mode when waked up. And enable CCM mode:

```
pwd_zcd=0x1
pwd_cmp_offset=0x0
loopctrl_en_rcscale=0x3
```

5.3.10. Enable and disable RBC bit

In Suspend mode, the RBC bit need to be set before executing WFI instruction and cleared after wake-up. The RBC bit is in CCM_CCR register. There is a VSTBY bit in CCM_CLPCR, it will change voltage to standby voltage (DCDC will enter power saving mode). This bit needs to be set in Low Power Idle mode and Suspend Mode. Enabling RBC_EN and VSTBY will ask analog to enter low power in STOP.

Two points need to be noted in enabling and disabling RBC_EN bit.

- GPC interrupts need to be masked before setting the bit.
- A delay (3us) is needed after setting the bit.

Example:

- Enable RBC_EN:

```
/* Mask all GPC interrupts before enabling the RBC counters to
       avoid the counter starting too early if an interrupt is already
       pending.
    */
    for (i = 0; i < LPM_GPC_IMR_NUM; i++)
    {
        gpcIMR[i] = GPC->IMR[i];
        GPC->IMR[i] = 0xFFFFFFFFU;
    }

    /* Enable the RBC bypass counter here to hold off the interrupts. RBC counter
     * = 32 (1ms). Minimum RBC delay should be 400us.
     */
    CCM->CCR = (CCM_CCR_RBC_EN_MASK | CCM_CCR_COSC_EN_MASK | CCM_CCR_OSCNT(0xAF));

    /* Recover all the GPC interrupts. */
    for (i = 0; i < LPM_GPC_IMR_NUM; i++)
    {
        GPC->IMR[i] = gpcIMR[i];
    }

    /* Now delay for a short while (3usec) Arm is at 528MHz at this point
     * so a short loop should be enough. This delay is required to ensure that
     * the RBC counter can start counting in case an interrupt is already pending
     * or in case an interrupt arrives just as Arm is about to assert DSM_request.
     */
    for (i = 0; i < 22 * 24; i++)
    {
        __NOP();
}
```

```
Disable RBC_EN:
/* Mask all GPC interrupts before disabling the RBC counters */
    for (i = 0; i < LPM_GPC_IMR_NUM; i++)
    {
        gpcIMR[i] = GPC->IMR[i];
        GPC->IMR[i] = 0xFFFFFFFFU;
    }
    /* Disable the RBC bypass counter */
    CCM->CCR &= ~CCM_CCR_RBC_EN_MASK;
    CCM->CCR &= ~CCM_CCR_REG_BYPASS_COUNT_MASK;

    /* Now delay for 2 CKIL cycles (61usec). Arm is at 528MHz at this point
     * so a short loop should be enough.
     */
    for (i = 0; i < 528 * 22; i++)
    {
        __NOP();
    }

    /* Recover all the GPC interrupts. */
    for (i = 0; i < LPM_GPC_IMR_NUM; i++)
    {
        GPC->IMR[i] = gpcIMR[i];
    }
```

5.3.11. Workaround for ERR007265

Errata ERR007265 describes about a case that SoC may fail to enter a low power mode.

ERR007265: CCM: When improper low-power sequence is used, the SoC enters low power mode before the Arm core executes WFI.

Software workaround:

1) Software should trigger GPR_IRQ to be always pending by setting IOMUX_GPR1_GINT.
2) Software should then unmask GPR_IRQ in GPC before setting CCM Low-Power mode.
3) Software should mask GPR_IRQ right after CCM Low-Power mode is set (set bits 0-1 of CCM_CLPCR).

5.3.12. How to run after suspend reset

5.3.12.1. Introduction

Suspend mode is a low power mode with the least power. But as the core power down, the wake-up is a reset wake-up. In System Idle and Low Power Idle mode, when wake-up the code can continue run after WFI instruction which is convenient but can’t get the least power consumption. The least power consumption can get under Suspend mode. After reset, the core can detect whether it is a suspend reset (via PGC->CPU_SR[PSR]). If

the system wants to run the code after WFI instruction, the restore data can be saved on OCRAM and the start address's offset is 32KB.That means FlexRAM0 cannot power down in Suspend mode.

5.3.12.2. SDRAM data storage suggestion

In most applications, user may want to use SDRAM to store data. SDRAM can enter self-refresh mode automatically in Suspend and data won't be lost.

CAUTION

Before entering Suspend mode, a STOP request should be sent to SDRAM to ask SDRAM to enter self-refresh mode. It is important to avoid accessing SDRAM and use variable in OCRAM.

6. Revision history

Revision number	Date	Substantive changes
0	11/2017	Initial release
1	01/2019	Updated the design of low-power solutions

7. Reference

1. i.MX RT 1050 Reference Manual
2. Arm Cortex M7 Reference Manual

.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: AN12085
Rev. 1
01/2019