Freescale Semiconductor
Application Note

Document Number: AN4503
Rev. 2, 04/2015

Power Management for Kinetis MCUs

When and how to use Kinetis low-power modes

Contents

1 Introduction

Applications strive for high performance within constrained energy budgets. Increasing system level requirements does not allow for compromises on performance, but pushes for low energy budgets to extend product use times. The Kinetis microcontroller family includes internal power management features that can be used to control the microcontroller's power usage and assist reaching the targets of embedded designs.

This application note discusses how to use the power management systems, provides use case examples, and shows real-time current measurement results for these use cases. A section on how to use the DDR memory controller to maintain the low power nature of your application is in this release.

Also included is a discussion of the differences between power management features on the various microcontrollers, along with drivers demonstrating these low-power states.Tips are given for using each of the power modes.

Power management methods discussed here do not include details on the clock generator module. Refer to the MCU reference manual for a description of the clock modes as well as the explanation of Multipurpose Clock Generator(MCG) acronyms. This application note focuses on the Power Management Controller (PMC), the Low Leakage Wakeup Unit (LLWU), the Reset Control Module (RCM), and the System Mode Controller (SMC).

For a one stop shop for all things low power on Kinetis refer to http://www.freescale.com/lowpower.

1.1 System Mode Controller Revisions - MC and SMC

The mode control is accomplished with a state machine in the MCU called the mode controller or system mode controller (SMC).

The code in this application note is compatible with Kinetis devices with the some differences. The mode controller module has evolved over time adding new low power modes and new run modes keeping mode entry and exit compatible. In this application note MC1 refers to the Mode Controller Module (MC) on Kinetis Rev 1.x 100 MHz devices. The terms MC2 - MC4 refer to the System Mode Controller modules (SMC) on subsequent Kinetis MCUs. With the introduction of the Kinetis L MC3(SMC) was created. Mode Controller version 4 was introduced with a reduced number of low power modes in the Kinetis E. With the introduction of Kinetis devices in 2014, including the K64, K22FN and KV, came MC5(SMC) with a high speed run power mode. See table below for summary.

Table 1. Kinetis Mode Controller Versions

Mode Controller version	Kinetis Series	Description
Mode Conctroller v1	K40,K60- 100 MHz	Kinetis Rev 1.x 100 MHz devices, with VBAT domain and low power OSC
SMC1 Mode Controller v2	Kinetis L 48 MHz	Added VLLS0 power modes No separate VBAT domain
SMC2 Mode controller v3	Kinetis K 72 MHz, K64/K24 120 MHz	Larger Memories M4 cores added LLS2, LLS3
SMC3 Mode Controller v4	Kinetis E	Only has RUN, WAIT, STOP
SMC4 Mode Controller v5	K65, K22FN, KV40 -some up to 180MHz	larger Flash and RAM some Include High Speed Run mode

1.2 Serialization of operation to guarantee correct sequence of operations

Consider the code segment below:

```
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
asm("WFI");
```

The ARM Cortex-M architecture does not delay the execution of the WFI instruction until the previous register write is completed. As a result, the core can assert the SLEEP output to the mode controller before the previous write to the mode controller register is complete. If this happens, the MCU may enter the wrong low-power mode.

The Kinetis MCU takes a minimum of 3 clocks to write to most peripherals. For peripherals like the SMC, that are powered in low power modes, it takes a minimum of 4 clocks for the data to be written to the SMC register. This is assuming no delay through the cross-bar. While the WFI asserts the SLEEP/SLEEPDEEP output in just one clock.

```
volatile unsigned int dummyread;
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;
SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x3);
dummyread = SMC_PMCTRL;   /* read-after-write sequence */
```

```
        dummyread++;   /* not required but added this statement to eliminate the compile
warning */
        asm("WFI");
```

The simplest (and guaranteed) way to avoid this problem is to follow the write to the control register, in this case the SMC_PMCTRL register, with a read of the same register. The read-after-write sequence guarantees that the write completes before the read, thus ensuring that the correct low-power mode is latched before the WFI instruction is executed.

This peripheral register read-after-write serialization sequence is applicable across all real-time sequence control. For the sake of consistency for software constructs, the peripheral register read-after-write sequence is a perfectly acceptable memory serialization approach for all control MCUs.

Warning: If the compiler optimization dial is turned in an attempt to improve code density, code sequences like this may get optimized out.

2 Power Management Techniques

2.1 Application Design Introduction

The design team coined the term "chasing nanowatts" to refer to the relentless process of going after every piece of circuitry in the chip, making sure the circuit was at the lowest power possible.

The following details just a few of the areas of focus during the application design process that resulted in the ultra low-power performance of the Kinetis MCUs.

The figure shows the idea of “reducing the area underneath the energy curve”.

In an embedded system where low power is critical it is all about being more energy efficient across the multiple tasks necessary to complete your application. These tasks can typically be summarized in 3 phases:

- Initialization phase
- Control phase which can include data collection, communication and control
- Compute phase

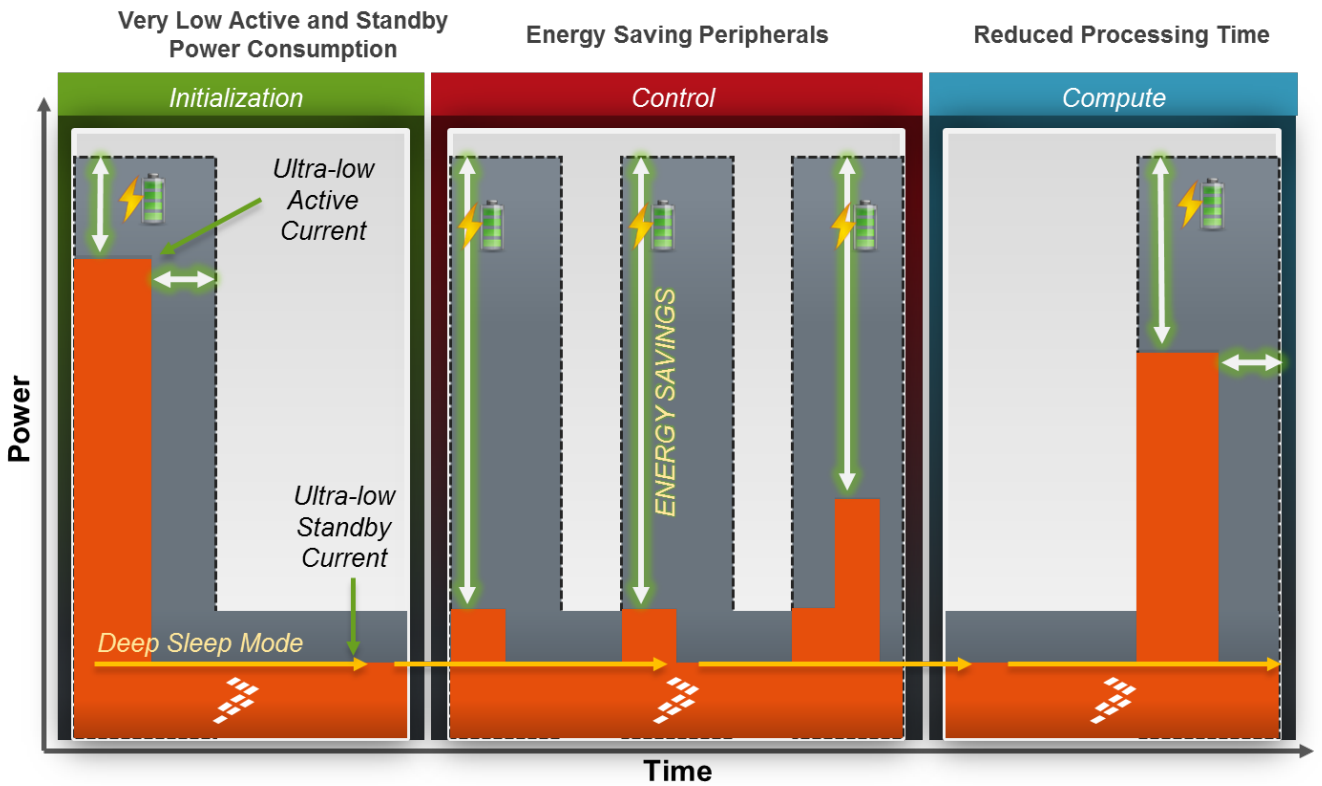


Figure 1. Energy Efficiency: Energy = Power x Time

To reduce energy across these phases the equation is simple; do more with less power and time (Energy = Power x Time). This is achieved by being low power in not just 1 phase, but in all. You must have very low active and standby power consumption.

The Kinetis MCUs can achieve both of these with RUN mode and a speed reduced Very Low Power RUN (VLPR) mode. Looking at the MCU data sheet (DS) “power consumption operating behaviors” section you will see specifications that can represent your use case. The test descriptions given in the DS show you the expected current when you use the controls to reduce the power consumed by your application. Just to name a few of these controls:

- frequency
- clock source
- voltage
- clock gating of peripherals
- execution from flash or RAM
- use of compute mode
- deep sleep modes

Use the DS nmbers as guides to your low power design efforts.

You can follow the graph in Figure 1 by seeing the energy savings comparing the Kinetis L and K2 Series energy curve in orange versus other competitive products energy curve in gray.

Second, you must have energy saving peripherals that are intelligent enough to collect, process, and store data without ever waking up the CPU. Other products would be required to wake-up into full RUN mode to activate a peripheral and complete the data collection phase later to return to a deep sleep mode.

For the Kinetis MCUs, the data collection phase starts in deep sleep mode and shows 3 periodic events triggered by a low power timer. The timer triggers the start of low power ADC conversions where the result is compared to a pre-programmed threshold value using built-in compare feature in ADC.

A feature like this avoids the need to store a result if the value is not within your desired result. Notice the first two events do not trigger the result to be stored. However, the last one does and instead of waking up the CPU to store the data, we have a much smaller energy spike since the Kinetis energy saving peripherals supports an asynchronous DMA wake-up feature that can store the ADC result to SRAM for later processing while the CPU is still sleeping. Once the DMA transfer is completed the MCU automatically returns to the deep sleep mode. Once you have collected sufficient data or transmitted the data using the low power UART you can then wake-up the CPU and begin the compute phase.

This is just one example usage of the available energy saving peripherals.

Last, you must reduce processing time in the compute phase in order to get back into deep sleep mode and start the whole sequence again. This is possible on Kinetis MCUs because of the ultra-efficient Cortex cores combined with an energy saving architecture including a cross bar allowing concurrent accesses between DMA and CPU to slave peripherals and a bit-manipulation engine simplifying bit oriented tasks and a flash memory controller eliminating wait states from flash accesses. The next few paragraphs will break down these concepts in a bit more detail.

2.2 Energy Saving Peripherals

Peripheral	*Low Power Functionality*
DMA	Allows energy-saving peripherals (ex. ADC, UART and Timer/PWM) to trigger asynchronous DMA request in STOP/VLPS modes to perform DMA transfer and return to current power mode with no CPU intervention
UART	Supports asynchronous transmit and receive operations to the bus clock supporting communication down to STOP/VLPS modes. Configurable receiver baud rate oversampling ratio from 4x to 32x allowing higher baud rates with lower clock sources
SPI	Supports slave mode address match wake-up function and first message capture down to STOP/VLPS modes
I2C	Supports multiple address match wake-up function down to STOP/VLPS modes
USB	Supports asynchronous wakeup on resume signaling down to STOP/VLPS
LPTPM (Timer/PWM)	Supports 16-bit timer input capture, output compare and PWM functions down to STOP/VLPS modes
LPTMR (Timer/Pulse Counter)	Supports 16-bit timer and pulse counter functions in all power modes
RTC	Supports 32-bit seconds counter with seconds interrupt and programmable alarm in all power modes with include temperature and voltage compensation

Figure 2. Energy Saving Peripherals 1

UART0 – can receive and transmit in VLPS mode. Can use DMA to receive data and store in a buffer or vice versa with the core shutdown. The address match allows characters to be received and ignored with no action being taken. For example, in say a networked smoke alarm system several nodes could be on the bus and a central system can individually address each detector to check status. Also, the UART supports a single wire system. Also, can receive the first character when in VLPS.

SPI and I2C support address match to wake-up functions. USB supports a wake-up on a resume signal.

LPTPM – supports timer, IC,OC and PWM down to VLPS. Useful for custom serial protocols. PWM output can be modified via DMA in VLPS in applications that provide a control voltage which may be updated occasionally from an ADC reading (light sensor controlling lighting brightness).

LPTMR provides for timer operation down to VLLS1 – wake up events and system “tick” function. Commonly used to trigger events (adc readings etc) or DMA transfers. Can also provide a pulse counting function down to VLLS1 – useful in flow meter applications, counting pulses from sensor, wake up x thousand pulses, otherwise always in VLLS1.

RTC – For devices without a dedicated RTC oscillator, this provides low power, less than 1uA, time keeping capability along with wake up capability down to VLLS1 with 32 KHz very low power crystal oscillator and down to VLLS0 if an external square wave clock is used.

Note that the asynchronous operation of the DMA and certain peripherals down to VLPS is now Kinetis L and new K device specific.

Peripheral	*Low Power Functionality*
ADC	Supports single conversions in multiple result registers down to STOP/VLPS modes with hardware averaging and automatic compare modes
CMP (Analog Comparator)	Supports threshold crossing detection in all power modes along with a triggered compare mode for lower average power compares
DAC	Supports static reference in all power modes
Segment LCD	Supports alternate displays and blink capability in all power modes
TSI (Capacitive Touch Sense Interface)	Supports wake-on capacitive touch on single channel in all power modes
LLWU (Low-Leakage Wake-up Unit)	Supports 8 wake-up pins, RESET and NMI wakeup pins, and some energy-saving peripherals in LLS and VLLSx modes

Figure 3. Energy Saving Peripherals 2

Continuing with the energy-saving peripherals the ADC supports single conversions with hardware averaging and auto compare modes. For example, in a thermostat, the MCU can be in low power mode with the LPTMR triggering an ADC reading regularly, only waking the MCU if the temperature exceeds some threshold.

The analog comparator has threshold crossing detection. It also has a trigger mode that allows the LPTMR to turn the CMP on for long enough to perform a comparison and then shut it down again.

The DAC supports a static reference. This can be updated in VLPS mode via DMA.

The segment LCD has built in blink mode and does not require CPU intervention and alternate blink mode that cuts in half the frequency of CPU intervention. These allow for power reduction. Operational down to VLLS1.

The capacitive touch sense interface supports wake-on touch from a single channel.

2.3 Architectural overview of power modes

The typical power modes in the legacy cores and other embedded systems are Run, Wait, and Stop. The ARM® Cortex™-M4 and M0+ power modes are Run, Sleep, and Deep Sleep. The extended power modes and their relationship to the typical and the ARM Cortex-M4 and M0+ cores are depicted in Figure 4.

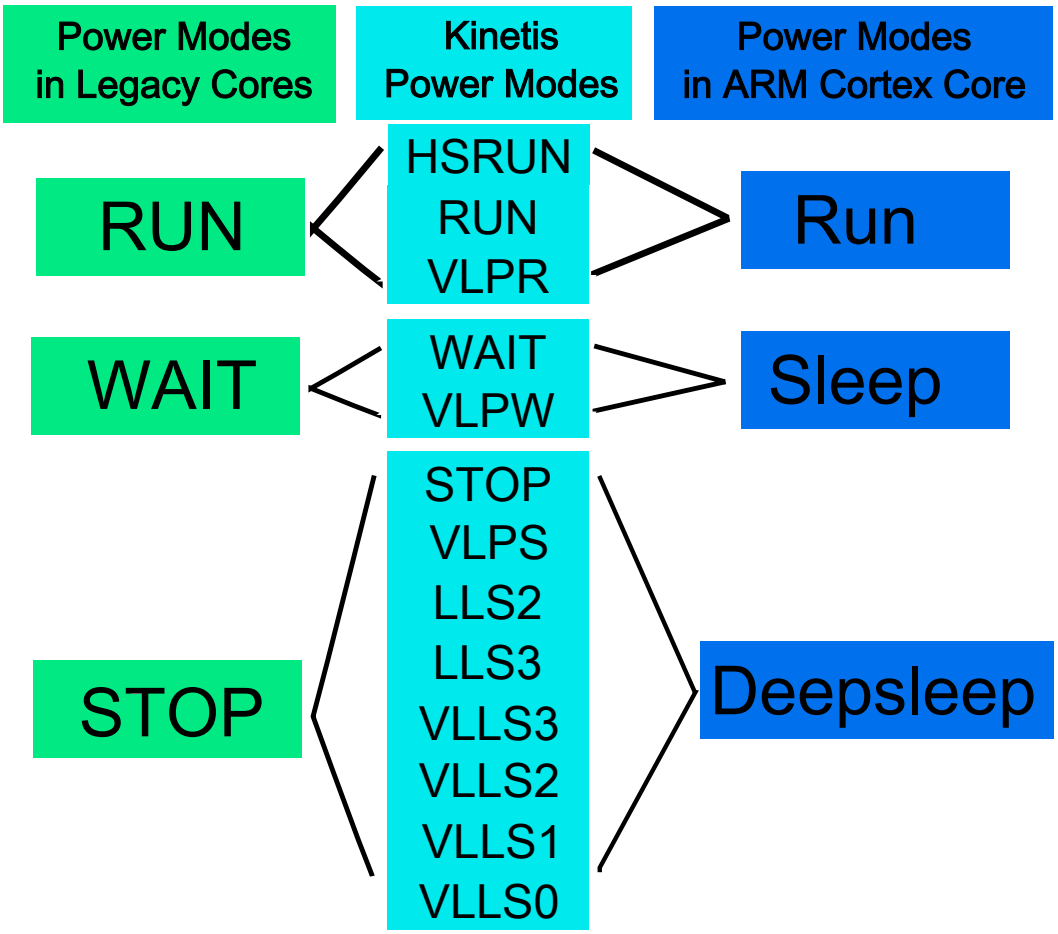


Figure 4. Power modes comparison

2.3.1 ARM Cortex-M4 and M0+ low-power implementation

The ARM Cortex-M4 and M0+ cores have three primary modes of operation: Run, Sleep, and Deep Sleep. On Kinetis, the cores use the wake from interrupt (WFI) instruction to invoke Sleep and Deep Sleep modes.

Figure 5 is a description of the Cortex -M4 and M0 low power implementation. The sleep and deep sleep states are architected in hardware to control the clock to the core and interrupt controller. When entering sleep the NVIC logic remains active and interrupts or a reset wake the core from sleep. When entering deep-sleep an Asynchronous Wakeup Interrupt Controller (AWIC) is used to wake the MCU from a select group of sources. These sources are described in the Low Leakage Wakeup Unit (LLWU) module.

Using the Sleep-On-Exit feature the ARM Cortex cores have one more way to enter low power modes. In the System Control Block of the Cortex processor is a register called the System Control Register (SCR) that contains several control bits related to sleep operation. Setting the control bit SLEEPONEXIT to 1 enables an interrupt driven application to avoid returning to a main application with every wake up event. When SLEEPONEXIT is enabled, the MCU can enter the low power mode as soon as it completes all pending exception handlers (interrupt service routines). It is just like execution a WFI immediately after the exception exit. The sleep-on-exit feature reduces unnecessary stack push and pop operations.

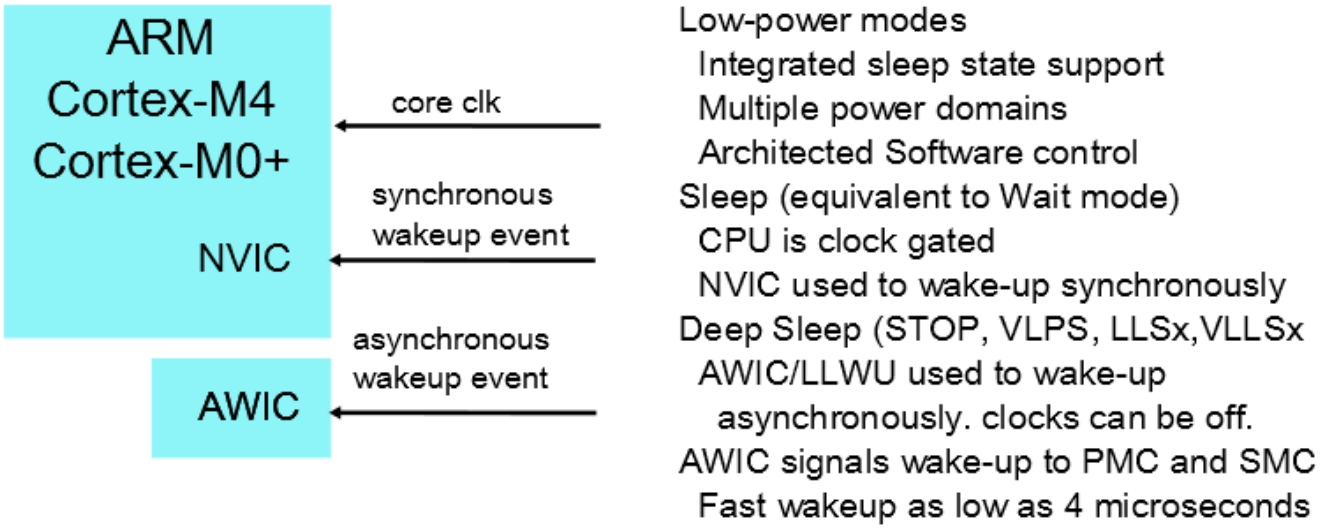


Figure 5. ARM Cortex-M4 and M0+ power modes

2.3.2 Mode description

Each of the power modes is described in Table 2 with details about the mode and the basics of mode entry and exit. The Kinetis MCU may have all of these low power modes or a subset. Some of the products revert to the traditional low power modes of Run, WAIT and STOP. Some do not have a LLWU module. See the MCU reference manual for the list of specific modes supported. For more advanced information, including what would prevent you from entering low-power mode, consult the reference manual power mode transitions section. The measurement data, frequencies, and other limit data given below are guidelines only. Make sure to use the MCU data sheet for the official values.

The following table describes the power modes available for most Kinetis devices. Refer to the individual device reference manual for details on the chip specific mode.

Table 2. Chip power modes

Chip mode	Description
Normal RUN	The MCU can be run at full speed and the internal supply is fully regulated, that is, in run regulation. This mode is also referred to as Normal Run mode.
High speed RUN	Allows maximum performance of the chip. The MCU can be run at a faster frequency compared with RUN mode and the internal supply is fully regulated. See the Power Management chapter for details about the maximum allowable frequencies.
Normal WAIT via WFI	Allows peripherals to function while the core is in sleep mode, reducing power. NVIC remains sensitive to interrupts; peripherals continue to be clocked. Run regulation is maintained.
Normal STOP via WFI	Places chip in static state. Lowest power mode that retains all registers while maintaining LVD protection. NVIC is disabled; AWIC is used to wake up from interrupt; peripheral clocks are stopped. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The internal voltage regulator is in run regulation.
VLPR (Very Low Power Run)	On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. Reduced frequency Flash access mode (1 MHz); LVD off; internal oscillator provides a low power 4 MHz source core, the bus, and the peripheral clocks.
VLPW (Very Low Power Wait) via WFI	Same as VLPR but with the core in sleep mode to further reduce power; NVIC remains sensitive to interrupts (FCLK = ON). On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency.

Table continues on the next page...

Table 2. Chip power modes (continued)

Chip mode	Description
VLPS Very Low Power Stop) via WFI	Places chip in static state with LVD operation off. Lowest power mode with ADC and pin interrupts functional. Peripheral clocks are stopped, but LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled (FCLK = OFF); AWIC is used to wake up from interrupt. On-chip voltage regulator is in a low power mode that supplies only enough power to run the chip at a reduced frequency. All SRAM is operating (content retained and I/O states held).
LLS (Low Leakage Stop)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. All SRAM is operating (content retained and I/O states held).
LLS3 (Low Leakage Stop3)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI,DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. All SRAM is operating (content retained and I/O states held).
LLS2 (Low Leakage Stop2)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. **NOTE:** The LLWU interrupt must not be masked by the interrupt controller to avoid a scenario where the system does not fully exit stop mode on an LLS recovery. A portion of SRAM_U remains powered on (content retained and I/O states held). RAM2 partition. The MCU is placed in a low leakage mode by reducing the voltage to internal logic and powering down the system RAM3 partition. The system RAM2 partition can be optionally retained using STOPCTRL[RAM2PO].The system RAM1 partition, internal logic and I/O states are retained.
VLLS3 (Very Low Leakage Stop3)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_U and SRAM_L remain powered on (content retained and I/O states held).
VLLS2 (Very Low Leakage Stop2)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. SRAM_L is powered off. A portion of SRAM_U remains powered on (content retained and I/O states held). The system RAM2 partition can be optionally retained using STOPCTRL[RAM2PO]. The system RAM1 partition contents are retained in this mode. Internal logic states are not retained.
VLLS1 (Very Low Leakage Stop1)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTimer, RTC, CMP, TSI, DAC can be used. NVIC is disabled; LLWU is used to wake up. All of SRAM_U and SRAM_L are powered off. The 32-byte system register file and the 128-byteVBAT register file remain powered for customer-critical data. I/O states are held. Internal logic states are not retained.
VLLS0 (Very Low Leakage Stop0)	Most peripherals are disabled (with clocks stopped), but LLWU and RTC can be used. NVIC is disabled; LLWU is used to wake up. All of SRAM_U and SRAM_L are powered off. The 32-byte system register file and the 128-byte VBAT register file remain powered for customer-critical data. I/O states are held. Internal logic states are not retained. The 1kHz LPO clock is disabled and the power on reset (POR) circuit can be optionally enabled using CTRL[PORPO].
BAT (backup battery only)	This chip is powered down except for the VBAT supply. The RTC and the 128-byte VBAT register file for customer-critical data remain powered.

2.3.3 Run mode

- Selected after any reset unless LPRUN is selected by an FOPT control.
- On-chip voltage regulator is On, full capability,unless LPRUN is selected by an FOPT control.
- Stack pointer (SP), Program Counter (PC), and link register are set.

- ARM Cortex-M4 and M0+ processor exits reset and reads the start SP.
- ARM Cortex-M4 and M0+ processor reads the start PC from vector table.
- If recovering from VLLSx exit through reset, OSC and output pins are held state until ACKISO written.
- If NMI is low at de-assertion of reset exception processing for the NMI vector is executed prior to instruction at start PC.

2.3.4 High Speed Run mode

- On-chip voltage regulator is in High-Speed RUN mode.
- Selected by a write to RUNM control bits.
- Flash programming and erasing is not allowed.
- Will exit HSRUN to run mode when RUNM bit written or Reset.
- Cannot enter low power modes without first moving to Run.

2.3.5 Very Low Power Run mode (VLPR)

- Can enter VLPR from RUN or from reset if LPBOOT bit in FOPT is set.
- On-chip voltage regulator is in a mode that supplies only enough power to run the MCU at a reduced frequency.
- Core and bus frequency limited to 4 MHz.
- Flash frequency limited to 800 kHz–1 MHz.
- Clock changes are not allowed.
- Flash programming and erasing is not allowed.
- Flex Memory (EEPROM) programming is not allowed.
- Can enter VLPR using the internal fast IRC, or LIRC if using the MCG-Lite.
- Can enter VLPR using an external clock.
- Can enter all low power modes directly except normal STOP and WAIT.
- Can exit VLPR via a RUNM control bit write or, on some devices, by means of an interrupt.

2.3.6 Wait mode

- ARM Cortex-M4 and M0+ core enters Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- NVIC remains sensitive to interrupts.
- Peripherals continue to be clocked.
- Reduce power by clearing clock gating bits in SCGCx registers.
- Flex memory (EEPROM) programming is not allowed.
- On interrupt, the ARM Cortex-M4 and M0+ core exits Sleep mode, Run mode processing resumes.
- Will exit WAIT in the same MCG mdoe present when WAIT was entered.

2.3.7 Very Low Power Wait mode (VLPW)

- Can only enter VLPW from VLPR.
- On-chip voltage regulator stays in a mode that supplies only enough power to run the MCU in a reduced frequency.
- ARM Cortex-M4 and M0+ core enters Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- NVIC remains sensitive to interrupts.
- Peripherals continue to be clocked.
- Reduce power by clearing clock gating bits in SCGCx registers allow individual module clock control.
- Flex memory (EEPROM) programming is not allowed.
- Flash memory programming/erase is not allowed.

- Will exit VLPW in the same MCG mode present when VLPW was entered.
- Can optionally keep the external reference clock enabled (up to 16 MHz max).
- On interrupt, the ARM Cortex-M4 and M0+ core exits Sleep mode, VLPR or Run mode processing resumes.

2.3.8 Normal Stop mode

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- Wakeup Interrupt Controller (WIC) is used to wake from interruptions.
- System and peripheral clocks are stopped unless the debug module is active. On some MCUs asynchronous clock available to low power modules and the DMA for operation in STOP.
- Low Voltage Detect(LVD) and Low Voltage Warning(LVW) are fully operational in STOP.
- All SRAM content retained. Register file, I/O and oscillator states held.
- Can enter Stop from any MCG mode.
- MCU can be configured to leave reference clocks running.
- Can optionally keep the PLL enabled but the output will be gated off—MCG_C5[PLLSTEN].
- Can optionally keep the internal reference clock enabled—MCG_C1[IREFSTEN].
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit Stop in the same clock mode when Stop was entered, except if Stop is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when Stop is exited.

MCU current will be higher when using a debugger since the ARM Cortex-M4 core will need to have clocks alive in Stop with the debugger enabled.

2.3.9 Very Low Power Stop mode (VLPS)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- WIC is used to wake from interruptions.
- System and peripheral clocks are stopped unless the debug module is active. On some MCUs asynchronous clock available to low power modules and the DMA for operation in VLPS.
- All SRAM content retained. Register file, I/O and oscillator states held.
- Can enter VLPS from any clock mode.
- MCU can be configured to leave reference clocks running.
- Can optionally keep the internal reference clock enabled—MCG_C1[IREFSTEN].
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit VLPS in the same clock mode when VLPS was entered, except if VLPS is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when VLPS is exited.

MCU current will be higher when using a debugger since the ARM Cortex-M4 core will need to have clocks alive in VLPS with the debugger enabled.

2.3.10 Low Leakage Stop modes (LLS ,LLS2 and LLS3)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled and interrupts need not be enabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped. OSCERCLK is optionally available for use by some peripherals.

- All SRAM (LLS3) or partial SRAM (LLS2) contents retained. Register File retained, I/O and oscillator states held.
- Most peripherals are in state retention mode (can't operate).
- Can enter LLS, LLS2 or LLS3 from any clock mode.
- MCU is static with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].
- Will exit LLSx in the same clock mode when LLSx was entered, except if LLSx is entered when in PLL Engaged External (PEE) mode and PLLSTEN = 0 the MCG will be in PLL Bypassed External (PBE) mode when LLSx is exited.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set.
- After taking the LLWU interrupt code, execution continues at the next instruction following the LLS mode entry.

2.3.11 Very Low Leakage Stop 3 mode (VLLS3)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled and interrupts need not be enabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS3. OSCERCLK is opntionally available for use by some peripherals.
- Most modules are disabled.
- All SRAM content retained. Register File retained, I/O and oscillator states held.
- Can enter VLLS3 from any clock mode.
- Upon a wake-up event, exit VLLS3 through the reset flow. MCU comes up in FLL Engaged Internal (FEI) mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from the reset vector.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.12 Very Low Leakage Stop 2 mode (VLLS2)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS2.
- Only a few modules are able to operate.
- 4–8K SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS2 from any clock mode.
- Can optionally keep the external reference clock enabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Upon a wake-up event, exit VLLS2 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from reset unless awakened by the NMI interrupt.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.13 Very Low Leakage Stop 1 mode (VLLS1)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.

- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS1.
- Only a few modules are able to operate.
- No SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS1 from any clock mode.
- Can optionally keep the external reference clock enabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- Upon a wake-up event, exit VLLS1 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from Reset.
- For Kinetis MCUs, the first enabled interrupt with the highest priority will execute.
- Can optionally keep the external reference clock enabled—OSC_CR[EREFSTEN].

2.3.14 Very Low Leakage Stop 0 mode (VLLS0)

- ARM Cortex-M4 and M0+ core enters Deep Sleep mode.
- ARM Cortex-M4 and M0+ core is clock gated.
- INTC/NVIC is disabled.
- LLWU must be configured to enable the desired wake-up source.
- System and peripheral clocks are stopped as in VLLS0.
- Power on Protection (POR) optionally enabled.
- No SRAM content retained, Register File retained, I/O and oscillator states held.
- Can enter VLLS0 from any clock mode.
- The Low power Oscillator (1 KHz) clock is off.
- The external reference clock is disabled.
- MCG is off with no clocks active (IREFSTEN and PLLSTEN have no effect).
- RTC can remain active when 32,768 Hz square wave input is provided to the MCU.
- Upon a wake-up event, exit VLLS0 through the reset flow. MCU comes up in FEI mode.
- Upon a wake-up event, the WAKEUP bit in the SRS register is set and the MCU executes the code from reset.
- The first enabled interrupt with the highest priority will execute.

2.4 Power Efficiency – Typical Power Consumption

Mode	K70 – 150 MHz	K20 – 50 MHz	KL46z – 48 MHz
Run	89.9 mA	13.9 mA	6.3 mA *
VLP Run (VLPR)	1.4 mA	867 uA – 1.1 mA	292 ** - 522 uA ***
Wait	40.9 / 19.6^ mA	7.5 mA	3.3 - 4.4mA
VLP Wait (VLPW)	926 uA	509 uA	261 uA
Stop	1.3 mA	310 uA	212 uA
VLP Stop (VLPS)	250 uA	3.5 uA	2.8 uA
LL Stop (LLS)	250 uA	2.1 uA	2.0 uA
VLL Stop 3 (VLLS3)	5.6 uA	2.9 uA	1.5 uA
VLL Stop 1 (VLLS1)	2.8 uA	1.5 uA	650 nA
VLL Stop 0 (VLLS0)	-	176 - 367 nA	130 - 330 nA

Cortex-M0+

* Compute Operation enabled: 4.1mA @ 48MHz core / 24MHz bus)
** Compute Operation enabled: 188uA @ 4MHz core / 800kHz bus)
*** Running Coremark algorithm, KEIL 4.54 optimized for speed
^Reduced Frequency Wait

Figure 6. Power Efficiency - Typical Power Consumption

Note these are all typical current estimates at 3V and 25C. RUN and VLPR numbers are with all modules off(compute Operation) clocking option enabled. In RUN mode at maximum core frequency of 48MHz the current consumed is 4mA. In VLPR mode at maximum core frequency of 4MHz the current consumed is 200uA. In both these modes with all modules off clocking option enabled the energy saving peripherals can still be functional.

2.5 Ultra Low Power Modes Check List

2.5.1 Items to do before entering low power mode

- The SMC_PMPROT must be set to allow the stop modes and low power run mode you wish to enter. This register is a write once after reset register. If no low power mode is allowed and SLEEPDEEP = 1, executing the WFI instruction enters Normal STOP low power mode.
- If using an external clock source then the clock monitor must be disabled before entering any low power mode except WAIT.
- A wake up source should be enabled before entering a low power mode. An interrupt for VLPS, WAIT, or VLPW and an LLWU source for LLSx, VLLSx modes. If none are enabled Reset and NMI will still wake the MCU from all low power modes..

- Select the desired low power mode in the SMC_PMCTRL and SMC_STOPCTRL registers.
- Set/Clear ARM SLEEPDEEP bit for STOP/WAIT.
- All of the steps above can be done in an initialization phase. If there are other operations, taking at least 12 core clock cycles, in-between the set of the stop mode and WFI instruction execution no serialization is needed.
- The Wait For Interrupt instruction, WFI, causes immediate entry to sleep mode unless the wake-up condition is true.
- If the low power mode is switched immediately before entering the sleep mode then you must ensure the write to the SMC_PMCTRL or SMC_STOPCTRL register has completed before executing the WFI. This can be done with a read of the SMC_PMCTRL register prior to execution of the WFI instruction. This is serialization.

2.5.2 Low power mode using sleep-on-exit:

- SCR register Sleep-On-Exit mode is set with SLEEPDEEP=1.
- Execute WFI to enter the first time.
- Normal STOP, VLPS or LLS can be used with Sleep-on-Exit operation.
- If the SLEEPONEXIT bit of the SCR is set to 1, when the processor completes the execution of all exception handlers it immediately re-enters sleep mode.
- Low power peripheral and asynchronous DMA can operate in VLPS with Sleep-On-Exit mode set.
- Use this mechanism in applications that require the core to run only when a wake up exception occurs.

2.5.2 To enter VLPR you must:

- Be in BLPI (using fast IRC for the MCG or LIRC is using MCG_Lite) or BLPE clock mode.
- Set the core frequency to 4MHz or less and flash clock to 1MHz or less.
- Not have the slow IRC enabled.
- Disable the clock monitor.
- Disable FIRC on devices using the MCG_Lite.
- Write the SMC_PMCTRL RUNM bits to enter VLPR.

2.5.4 Exiting VLPR

- By Writing the SMC_PMCTRL RUNM bits to exit VLPR
- On K series with an interrupt.
- Exiting VLPR with a write to the RUNM bits does not change the flow of the code. No exception is generated.

2.5.5 Exiting LLS, LLS2, or LLS3

- By a low leakage wake up (LLWU) event, LLWU interrupt is pending after wake up and takes priority.
- Internal module flags can be cleared in the LLWU exception handlers (ISR) or if you leave the module flag set then after the LLWU ISR is complete you enter the module ISR to clear the flag. If you clear the module flag and do not want to enter the module ISR the NVIC clear pending interrupt flag bit must be cleared for the module. Serialization of operations is necessary to ensure the flag are cleared before returning from the LLWU or module ISRs.
- Code re-starts at the instruction following the WFI instruction after completing of all exception handlers.

2.5.6 Exiting VLLS3, VLLS2, VLLS1, or VLLS0

- VLLSx recovery is via the reset flow.
- After exiting any of the VLLSx modes, the IO remain latched until they are released by writing “1” to the ACKISO bit in the PMC.
- The GPIO must be re-configured before ACKISO is written.

- If there is no requirement to make use of any external IO, There is no need to re-configure the IO before re-entering VLLSx if you do not write the ACKISO bit.
- If the oscillator has been kept running in VLLS1,VLLS2, or VLLS3, it must be re-configured before ACKISO is written (unless it was configured in RTC OSC).
- RAM contents need to be re-initialized if using VLLS1 or VLLS0.
- RAM contents need to be re-initialized for RAMs not powered in VLLS2.
- The LLWU interrupt will be pending.
- If the wake up was by a module then the associated Module interrupt will also be pending.
- The interrupt taken will be the first one enabled in the reset flow.
- If the LLWU interrupt is the first enabled then the module interrupt flag must be cleared in the LLWU ISR.
- If the Module interrupt is the first enabled then the module ISR will clear the interrupt flag and the LLWU ISR will no longer be pending.
- Pin wake up flags must be cleared in the LLWU.
- VLLS wakeup will be reported in the RCM.

NOTE

VLLS0 mode is only available on some Kinetis devices.

2.6 Kinetis Energy Savings – Tips and Tricks

1. The absolute lowest power mode is "powered VDD OFF". This mode uses the RTC_WAKEUP to control the MCU VDD. On some later Kinetis devices with isolated VBAT power domains, like the K22F, K64, K24 and K65, there is a control output called RTC_WAKEUP_B that can be used to gate on the MCU VDD using an external component.
2. Use the FOPT options: There are multiple FOPT register bits such as LPBOOT to help reduce current at startup or disable an unwanted function like EZPORT mode entry.
3. Using the asynchronous DMA, in conjunction with a low power peripheral is technically another low power mode. While in the low power mode the peripheral, such as the LPUART, can send and receive data until buffers need tending. In STOP or VLPS with Core clock disabled:
    - UART data transfers still possible
    - PWM output changes possible with DMA
    - ADC data sampling possible with DMA reading result register
4. UART0/LPUART – Low Power UART:
    - Over-Sampling, Address Match, Asynchronous operation in conjunction with asynchronous DMA.
5. Low Power Timer (LPTMR) running off of the LPO offers a low resolution(typical 1ms) wakeup timer in all but the lowest power mode VLLS0.
    - Very low current added in all power modes.
    - Operates through all resets except POR and LVD.
    - Can run from the internal 1 KHz low power oscillator in all power modes except VLLS0.
6. Real Time Clock (RTC) running off an external square wave clock source is the lowest power option for a periodic wake up source.
    - Lowest current adder for a wakeup source.
    - Can operate in all power modes
    - Operates through all resets except POR and LVD.
    - Operates from external 32,768 Hz low power crystal oscillator in all power modes except VLLS0.
7. Unused pins should be configured in the disabled state, mux(0), to prevent unwanted leakage (potentially caused by floating inputs).
8. Turn off unnecessary module clock gates in SIM.
9. Use compiler options to reduce the current spikes from flash access for the literal pool. The literal pool is used as the base address of a module's register bank. The compiler could optimize the literal pool address.
10. Keep loops as small as possible to keep the code access within the size of the cache.
11. Execute commonly used loops or functions out of RAM instead of FLASH.
12. Use explicit register writes rather than Read-Modify-Writes.

13. ADC: Use 4x h/w averaging if 32x is not needed.
14. Use lower power mode with a slower clock speed if possible.
15. For GPIO accesses use the IOPORT where available. It takes less clock cycles to propagate the write to the output pin.
16. For battery applications with USB capability, add an isolation diodes in the supply line and use the USB to power the application when connected to the USB bus (can use on chip regulator).
17. Also for battery applications with USB capability, use the VBUS presences to wakeup the MCU for low power modes by connecting the VBUS through appropriately sized divide resistors to an LLWU wake-up input pin.
18. Compiler optimization levels can affect Idd.
19. Lowest Idd is not necessarily at the lowest power. Figure 7 shows a typical IDD versus Voltage curve for a Kinetis L MCU. Figure 8 below shows the overall power curve power = IDD * VDD. Notice that running the MCU at lower VDD values does reduce power consumption but at the expense of higher IDD's.

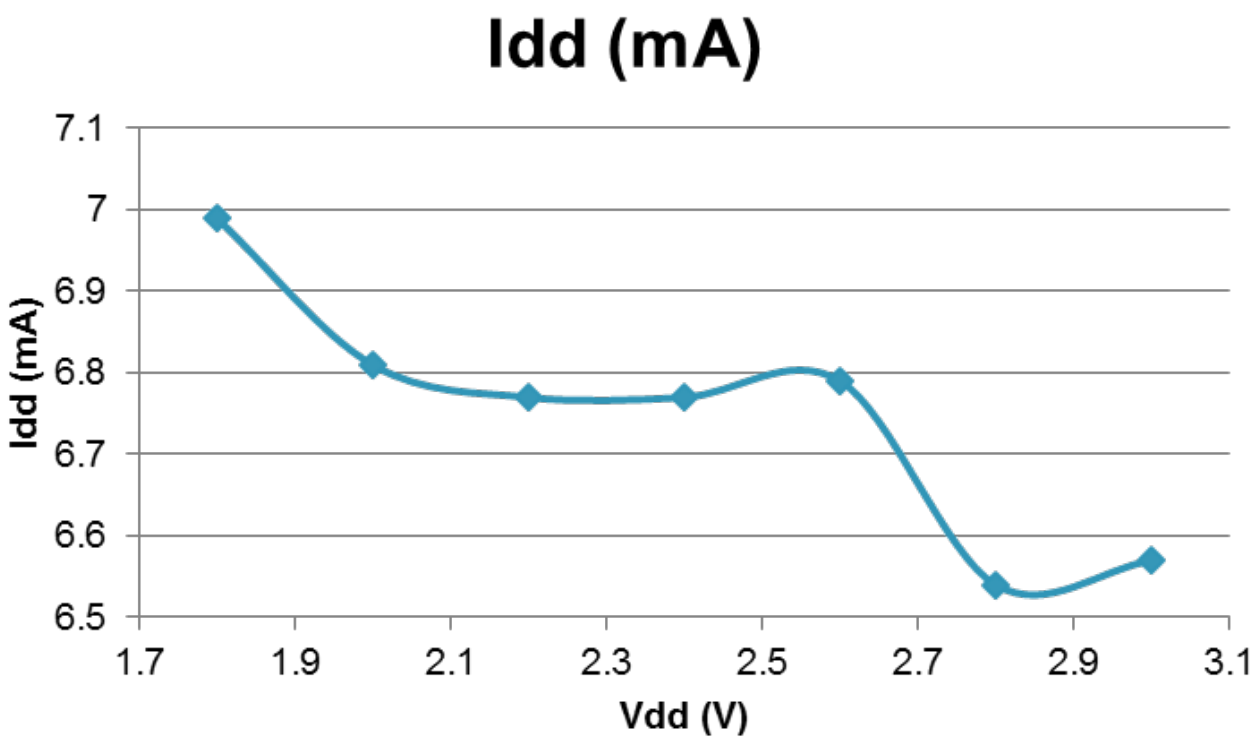


Figure 7. Kinetis MCUs typical Idd versus Vdd

Lowest Power is achieved at lowest voltage

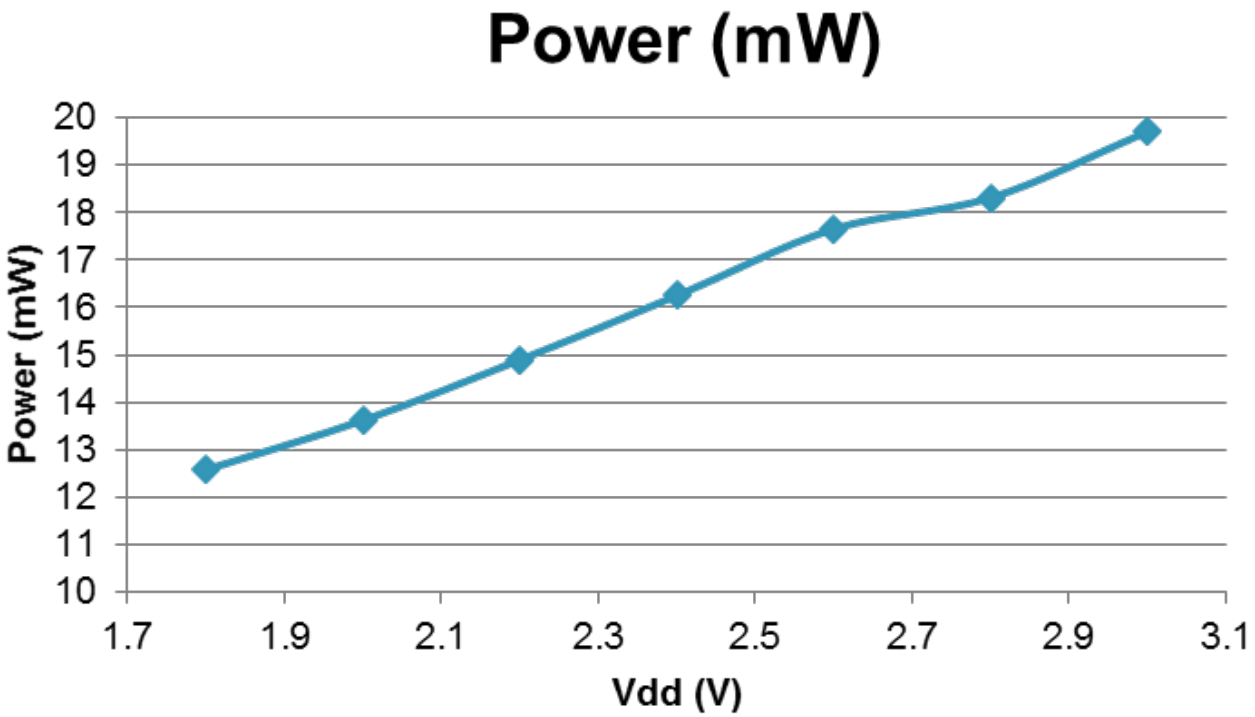


Figure 8. Kinetis MCUs Power versus Vdd

3 Quick Start Kinetis SDK

With the release of the Kinetis SDK, a growing number of Kinetis MCUs are supported with low-level software to set up and operate the MCU. The lowest level code in the SDK is the Hardware Abstraction Layer (HAL). The next level up from the HAL is the reference peripheral drivers and system services drivers. Power mode control is part of the system services driver set and allows for advanced device management when entering and exiting power modes, whereas the HAL functions allow you create your own customized driver and offer more flexibility.

The SDK release contains demo projects that provide use-case based examples that you can re-use for you own application. One such demo is the power_manager_demo. This code provides you a working power mode transition example for your study and re-use.

This quick start section demonstrates how you can use either the HAL or the system services to put the Kinetis MCU into the LLS low power mode from RUN mode. If your application began from a non-SDK based project or started as an SDK HAL based project, you will likely want to continue using the HAL (and the HAL example will assume this). Otherwise, if your project began from an SDK driver or system services based application, you will probably want to continue using the reference peripheral drivers (and the power manager system service example will assume this). First we will focus on using the HAL layer and then we will focus on using the prebuilt drivers.

3.1 LLS mode entry and exit - Kinetis Software Development Kit (KSDK) code example

3.1.1 MCU preparation using the SDK HAL functions

The first step in moving to a low power mode is to prepare the MCU for that low power mode. This includes turning off any peripheral modules that are not needed in the desired mode and any unused GPIO pins should be put in the disabled state. The SDK HAL provides easy to use functions to allow for this. An example of one such statement is shown below.

```
/* Disable PTC7 */
PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );
```

This should be done for every pin that is used in the application.

3.1.2 Wakeup source configuration

Then, the exit method must be configured. This example uses a push button connected to a GPIO pin for the exit method. The first step in configuring the exit method is to initialize the GPIO pin connected to the push button. The code below demonstrates this using the KSDK HAL functions. Specifically, the following code will:

- **Configure a GPIO pin that will be used to wake from LLS mode**.
- **Configure pin as digital input**
- **Configure LLWU module pin PORTC6 (LLWU_P10) as a valid wake-up source**.

```
void main (void){
  /* Enable Port C6 to be a digital pin.   */
  SIM_HAL_EnablePortClock(SIM_BASE, HW_PORTC);

  /* PORTC_PCR6 */
  PORT_HAL_SetMuxMode(PORTC_BASE,6u,kPortMuxAsGpio);

  /* Configure GPIO input features. */
  PORT_HAL_SetPullCmd(PORTC_BASE, 6u, true);
  PORT_HAL_SetPullMode(PORTC_BASE, 6u, kPortPullUp);

  /* Set the LLWU pin enable bits to enable the PORTC6 input
  * to be a wake-up source.
  * WUPE10 is used in this case since it is associated with PTC6.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_HAL_EnableLlwuClock(SIM_BASE, HW_LLWU); // this bit may not exist on some MCUs
  LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);

  //falling edge detection
  LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, kLlwuExternalPinFallingEdge, kLlwuWakeupPin10);
```

3.1.3 SMC configuration using the SDK HAL functions

After preparing the MCU peripherals for entry into low power mode and configuring the wakeup source, it is time to configure the SMC (or MC depending on your specific MCU). Follow these steps:

- First, the power mode protection register must be configured to allow entry into the desired mode. This is a write once register and can only be written once after every reset. The recommended location for writing this register is in your startup code.

  ```
  /* Power mode protection initialization */
  SMC_HAL_SetProtection(SMC, kAllowPowerModeLls);
  ```

NOTE

Multiple power modes may be enabled by simply OR'ing the appropriate masks together in the function call. For example, if VLPS mode were to be allowed as well, the function call would become SMC_HAL_SetProtection(SMC, kAllowPowerModeLls | kAllowPowerModeVlps);

- Next the stop mode, and sub-mode must be configured. Once these are configured, the sleep bit in the ARM System Control Block (SCB) should be set and then the Wait For Interrupt (WFI) instruction should be executed. The SDK HAL provides a convenient way of performing these tasks in a single function call. This is the SMC_HAL_SetMode function and an example of its use is shown below:

```
/* First, setup an smc_power_mode_config_t variable to be passed to the SMC
 * HAL function
 */
smc_power_mode_config_t smc_power_config = {
    .powerModeName = kPowerModeLls,          // Set the desired power mode
    .stopSubMode = kSmcStopSub3,             // Set the sub-mode if necessary
#if FSL_FEATURE_SMC_HAS_LPWUI
    .lpwuiOption = false,                   // Set the LPWUI option if available
    .lpwuiOptionValue = kSmcLpwuiDisabled,
#endif
#if FSL_FEATURE_SMC_HAS_PORPO
    .porOption = false,                  // Set the PORPO option.  Only needed
    .porOptionValue = kSmcPorEnabled,   //  if entering VLLS0
#endif
#if FSL_FEATURE_SMC_HAS_PSTOPO
    .pstopOption = false,               // Only needed if entering a PSTOP mode
    .pstopOptionValue = kSmcPstopStop,
#endif
};

/* Disable clock monitor before moving to a STOP mode */
CLOCK_HAL_DisableOsc0Monitor(MCG_BASE);

/* Now call the SMC HAL SetMode function to move to the desired power mode. */
SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
```

NOTE

The SMC_HAL_SetMode function simply configures the SMC control registers and executes the WFI instruction and performs the same actions every time regardless of whether or not the registers need to be written. If your application calls for more advanced functions, such as faster low power entry times or sleep-on-exit usage, there are ways to accomplish these tasks. They are presented here:

- Faster low power entry times: Once the SMC_HAL_SetMode function has been called, the SMC's control registers are still in the same configuration once the function exits. So if the same low power mode is desired on subsequent entries, you can simply execute the WFI instruction as shown:

  ```
  __WFI();
  ```

- Sleep-on-exit: To use the sleep on exit feature, simply set the sleep-on-exit bit in the System Control Register (SCB->SCR) before calling the SMC_HAL_SetMode function. The SMC_HAL_SetMode function performs only read-modify write operations on the SCB->SCR register so the configuration of the sleep-on-exit bit will be preserved. An example of how to do this is shown below:

  ```
  /* Set the Sleep-on-Exit bit */
  SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;

  /* Then call the SMC_HAL_SetMode function */
  SMC_HAL_SetMode(SMC_BASE, &smc_power_config);
  ```

3.1.4 Exiting Low Leakage Stop (LLS) low-power mode

The MCU is now in LLS mode. The wakeup pin must be asserted to wake the MCU from LLS (in this case, a falling edge on Port C pin 6). At a minimum, the following code must be executed to clear the wake-up event flag in the LLWU register (note that for LLS or VLLSx modes, interrupts do not need to be enabled but can be to simplify your LLS or VLLS exit procedure).

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_HAL_GetExternalPinWakeupFlag(LLWU_BASE, 10u)) {
      LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, 10u);
  }
} /* end of main */
```

NOTE

If interrupts are enabled, the execution resumes with the appropriate interrupt service routine. If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.

3.1.5 MCU preparation using the power manager system service

MCU preparation for low power entry is the same using the power manager system service as it is when using the HAL with one difference. Any peripheral module that is not needed in the desired mode should still be turned off and any GPIO pin that will not be used should still be put in the disabled state. The difference is that a callback function can be used. An example of a callback function to do this is as follows:

```
/* Structure for callback data */
typedef struct {
    uint32_t counter;
    uint32_t status;
    uint32_t err;
} callback_data_t;

/* Callback structure definition to be used by the Power Manager variables */
typedef struct {
    callback_data_t none;
    callback_data_t before;
    callback_data_t after;
    power_manager_callback_type_t lastType;
    uint32_t err;
} user_callback_data_t;

/* This is the declaration of the callback function that can be used
     * in power mode transitions
     */
    user_callback_data_t callbackData0;

    power_manager_static_callback_user_config_t callbackCfg0 = { callback0,
            kPowerManagerCallbackBeforeAfter,
            (power_manager_callback_data_t*) &callbackData0 };

    power_manager_static_callback_user_config_t *callbacks[] =
            { &callbackCfg0 };

/*
 * Power manager callback implementation code
 *
 * This section defines what the power manager functions do before
```

```
 * and after the power manager mode transitions occur.  It also defines
 * what to do if no callback is provided.
 *
 */
power_manager_error_code_t callback0(power_manager_callback_type_t type,
        power_manager_user_config_t * configPtr,
        power_manager_callback_data_t * dataPtr) {

    user_callback_data_t * userData = (user_callback_data_t*) dataPtr;
    power_manager_error_code_t ret = kPowerManagerError;

    switch (type) {

    case kPowerManagerCallbackNone:

        userData->none.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackBefore:

        /* Disable PTC7 and any other pins that are not used */
        PORT_HAL_SetMuxMode(PORTC,7u,kPortPinDisabled );

        userData->before.counter++;
        ret = kPowerManagerSuccess;
        break;
    case kPowerManagerCallbackAfter:

        userData->after.counter++;
        ret = kPowerManagerSuccess;
        break;
    default:
        userData->err++;
        break;
    }

    userData->lastType = type;

    return ret;
}
```

3.1.6 Wakeup source configuration using the power manager system service

The first step in configuring a wakeup source is adding the desired pin in the _gpio_pins enumeration. The code below is an example of this:

```
/*! @brief gpio pin names.*/
/*!*/
/*! This should be defined according to board setting.*/
enum _gpio_pins
{
    kGpioSW1         = GPIO_MAKE_PIN(GPIOC_IDX, 6),  /* TWR-K22F120M SW1 */
};
```

Next, the pin must be included in an input pin structure that defines all of the necessary parameters for that pin to work in your application. This structure is used by the GPIO driver to properly initialize the desired pin:

```
  /* Declare Switch pins */
gpio_input_pin_user_config_t switchPins[] = {
    {
        .pinName = kGpioSW1,
        .config.isPullEnable = true,
        .config.pullSelect = kPortPullUp,
        .config.isPassiveFilterEnabled = false,
        .config.interrupt = kPortIntDisabled,
    },
```

```
    {
        .pinName = GPIO_PINS_OUT_OF_RANGE,
    }
};
```

Another important action in enabling a GPIO pin as a low-power wakeup source is to configure the port mux field correctly. This is done in the hardware_init function in hardware_init.c. It is also important to only enable those pins that will be used (for minimum current draw). An example of how to enable a pin for a GPIO function is presented below.

```
/* enable clock for PORTC */
CLOCK_SYS_EnablePortClock(2);

/* enable PORTC pin 6 as a GPIO */
PORT_HAL_SetMuxMode(PORTC,6u,kPortMuxAsGpio);
```

Now the GPIO driver may be called to initialize the desired pins.

```
 /* Initialize the pins used for switches and LEDs only */
    GPIO_DRV_Init(switchPins, NULL);
```

Next, since the low-leakage power modes (LLS, VLLSx) use the Low-Leakage Wakeup Unit (LLWU) as the recovery method, the LLWU must be configured. To do this, we use the LLWU HAL functions as shown below.

```
/*************************************************************
     * LLWU pin initialization
     *
     * First, clear the associated flag just to be sure the device
     * doesn't immediately enter the LLWU interrupt service routine
     * (ISR). Then enable the interrupt
     *************************************************************/
    LLWU_HAL_ClearExternalPinWakeupFlag(LLWU_BASE, pinEn);

    LLWU_HAL_SetExternalInputPinMode(LLWU_BASE, riseFall, pinEn);
```

In the function calls above, LLWU_BASE is the register location of the LLWU (which is defined in the device specific header file), risefall is a variable of type llwu_external_pin_modes_t (which is defined in fsl_llwu_hal.h), and pinEn is a uint8_t variable that specifies which pin in the LLWU to enable.

3.1.7 Entering Low Leakage Stop (LLS) low-power mode using the power manager system service

The Kinetis SDK power manager system service utilizes an array of power configurations to interface to the application. The user is expected to create an array of all of the power configurations to be used in the application. An example of an array definition to be used by the power manager is shown below.

```
    /* Power manager configuration variables that configure
     * the named low power mode
     */
    power_manager_user_config_t llsConfig;
    power_manager_user_config_t runConfig;

    /* This is the power manager configuration variable.  The order of this
     * list determines the index that should be passed to the Setmode
     * function to enter the desired power mode.
     */
    power_manager_user_config_t const *powerConfigs[] = { &llsConfig, &runConfig };
```

After declaring the power manager user configurations and defining the array for the power manager function calls, the power manager configuration variables must be configured. An example is shown below.

```
    /* Define the power mode configurations */
    /* Define all of the parameters for the LLS configuration */
    llsConfig.mode = kPowerManagerLls;
    llsConfig.policy = kPowerManagerPolicyAgreement;
```

```
#if FSL_FEATURE_SMC_HAS_LPWUI
    llsConfig.lowPowerWakeUpOnInterruptOption = true;
    llsConfig.lowPowerWakeUpOnInterruptValue = kSmcLpwuiEnabled;
#endif
    llsConfig.sleepOnExitValue = false;
    llsConfig.sleepOnExitOption = false;
#if FSL_FEATURE_SMC_HAS_PORPO
    llsConfig.powerOnResetDetectionOption = true;
    llsConfig.powerOnResetDetectionValue = kSmcPorEnabled;
#endif
#if FSL_FEATURE_SMC_HAS_LPOPO
    llsConfig.lowPowerOscillatorOption = true;
    llsConfig.lowPowerOscillatorValue = kSmcLpoEnabled;
#endif
```

NOTE

If the device you are working with has LLS submodes (i.e., LL2 and LL3) then you will want to specify that by using the *.mode variables kPowerManagerLls2 and kPowerManagerLls3.

After configuring the power manager user configuration variables, the power manager needs to be initialized and the callbacks need to be registered. The following code is an example of how to do this. .

```
/* Initialize the power manager module */
POWER_SYS_Init(&powerConfigs,
     sizeof(powerConfigs)/sizeof(power_manager_user_config_t *),
     &callbacks,
     sizeof(callbacks)/sizeof(power_manager_callback_user_config_t *));
```

The above function simply informs the power manager system service of what configurations are available. To use the drivers to move to the low power mode, use the POWER_SYS_SetMode function and pass the index to the configuration in the powerConfigs array defined in the code above. This function returns a status variable to acknowledge that the mode change occurred successfully or unsuccessfully (note that this code will not be executed if moving to a VLLSx mode, as these modes are exited through the reset sequence. The following is an example.

```
/* Now Issue power mode change */
ret = POWER_SYS_SetMode(powerModeLls1Index);

/* The code execution should not get here on successful entries.
 * However, unsuccessful entries should make it to this code and
 * report errors
 */
if (ret != kPowerManagerSuccess)
{
    printf("POWER_SYS_SetMode(powerModeLls1Index) returns : %u\n\r", ret);
}
```

To explore these drivers further, please refer to the power_manager_hal_demo in your SDK installation found at <SDK root>/apps/<board name>/demos/power_manager_hal_demo.

4 Quick Start

4.1 LLS mode entry and exit - bare metal code example

For those who want to quickly see low-power operation, simply add the following lines of code to enter Low Leakage Stop (LLS) mode. To enable a wake-up pin, follow the steps in the initialization section below. The initialization section also disables the clock monitor, which is needed only if the application is running from an external clock source.

I chose to demonstrate LLS mode since it is one of the more useful low power mode. It exits the low power mode on the next instruction following the WFI instruction that put the MCU into the low power mode. LLS is very fast to wakeup, as quickly at 4 us in FLL engaged internal (FEI) clock mode at 100 MHz. It is the first mode in the sequence of low power modes to need the LLWU to wake up the MCU from low power mode. LLS mode maintains all register and SRAM though the low power mode.

4.1.1 Initialization

The code below will:

- **Configure a GPIO pin that will be used to wake from LLS mode**.
- **Configure pin as digital input** (Kinetis device shown)
- **Configure LLWU module pin PORTE1 (LLWU_P0) as a valid wake-up source**.

```
void main (void){
  unsigned int dummyread;
  /* Enable Port E1 to be a digital pin.   */
  SIM_SCGC5 = SIM_SCGC5_PORTE_MASK;
  PORTE_PCR1 =(PORT_PCR_ISF_MASK |  //clear flag if there
              PORT_PCR_MUX(01) |   //set pin functionality -GPIO
              PORT_PCR_IRQC(0x0A)| //falling edge interrupt enable
              PORT_PCR_PE_MASK |   // pull enable
              PORT_PCR_PS_MASK);   // pullup enable

  /* Set the LLWU pin enable bits to enable the PORTE1 input
  * to be a wake-up source.
  * WUPE0 is used in this case since it is associated with PTE1.
  * This information is in the Chip Configuration chapter of the Reference Manual.
  * 0b10: External input pin enabled with falling edge detection
  */
  SIM_SCGC4 = SIM_SCGC4_LLWU_MASK; // this bit may not exist on some MCUs
  LLWU_PE1 = LLWU_PE1_WUPE0(2); //falling edge detection
```

- **If an external clock source is used, disable the clock monitor(s)**

```
  /* Need to make sure the clock monitor(s) is disabled if using
  * an external clock source. This assumes OSC0 is used as the
  * external clock source and that the clock gate for MCG has
  * already been enabled.
  */
  MCG_C6 &= ~MCG_C6_CME0_MASK; //CME=0 clock monitor disable
```

4.1.2 Entering Low Leakage Stop (LLS) low-power mode

The code below will:

- **Allow the MCU to go into LLS mode. The PMPROT register is a write-once register and should be written to allow all needed low-power modes. If only LLS and VLLS1 are needed, then enable only those modes.**

```
/* Write to PMPROT to allow LLS power modes */
  #ifdef MC1
  MC_PMPROT = MC_PMPROT_ALLS_MASK; // (MC1)
  #else
  SMC_PMPROT = SMC_PMPROT_ALLS_MASK;   // (MC2)
  #endif
```

- **Set the LPLLSM (for MC1) or STOPM (for MC2) bits to select the LLS low-power mode. If VLLS1 mode is desired there is one more write for MC2 MCUs.**
- **Set the VLLSM bits to 0b11 for LLS3. If LLS2 with reduced SRAM retention is desired choose 0b10.**

```
#ifdef MC1
MC_PMCTRL  =  MC_PMCTRL_LPLLSM(3); // (MC1) set LPLLSM = 0b11
#else
SMC_PMCTRL =  SMC_PMCTRL_STOPM(3); // (MC2) Set STOPM = 0b11
#endif

#ifdef MC4
SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);  // (MC4) Set VLLSM = 0b11 for LLS3
#endif
```

- **For Kinetis devices, set the SLEEPDEEP bit in the core, and then execute the Wake From Interrupt (WFI) instruction to enter LLS mode.**
- **Serialization read of the PMCTRL register is needed to ensure that the core does not stop operation before the write to set the low-power mode is completed. See Power mode transition code section for details on serialization.**

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

/* Set the SLEEPDEEP bit to enable deep sleep mode */
SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

/* WFI instruction will start entry into low-power mode */
asm("WFI");
```

- **Serialization read of the PMCTRL register is needed to ensure that the core does not stop operation before the write to set the low-power mode is completed. See Power mode transition code section for details on serialization.**

```
/*wait for write to complete to SMC before stopping core */
dummyread = SMC_PMCTRL;

SIM_SOPT4 = SIM_SOPT4_STOPE_MASK;  //enable stop mode entry
asm (  stop #0x2000; )
```

4.1.3 Exiting Low Leakage Stop (LLS) low-power mode

- **The MCU is now in LLS mode.**
- **The LLWU wake-up circuitry operates independently of interrupts.**
- **Interrupts do not need to be enabled nor does the LLWU interrupt vector need to be enabled for the MCU to wake from LLS or VLLSx low-power modes.**
- **If a falling edge event occurs on PORTE1, the MCU will exit LLS and return to Run mode.**
- **After exiting from LLS, the code below clears the wake-up event flag in the LLWU flag register.**
- **If interrupts and the LLWU interrupt are enabled, execution resumes with the LLWU interrupt service routine. Then code execution returns to the instruction following the WFI or STOP instruction.**
- **If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.**

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
      LLWU_F1 |= LLWU_F1_WUF0_MASK;
  }
} /* end of main */
```

5 Reset Management

5.1 Resets and holds on I/O for low-power modes

Managing resets in the Kinetis MCU has become more complicated than previous microcontrollers with the introduction of the new low-power modes that recover through the reset flow. A reset is not a simple “return all modules to a default state” function anymore either. Many of the modules that are used to wake the MCU from low power modes will not clear the register state with certain reset types.

In the Reset section is an extended discussion on all of the types of reset. Here is a short list of Reset types:

1. Power On Reset (POR) and Low Voltage Detect (LVD) resets that occur when power is initially applied to the MCU. With multiple power domains there is a POR reset for both the VDD power domain and the VBAT power domain. A POR occurance in the VBAT can only be detected with the RTC invalid flag.
2. External pin reset (PIN) This pin is open drain and has an internal pullup device. Asserting RESET wakes the device from any mode. If waking from a VLLSx low power mode the hold on the I/O pins is disabled.
3. Computer operating properly (COP) watchdog timer
4. Technically not a reset source but if waking from VLLSx low power mode the LLWU Low leakage wakeup Unit does result in a reset. The (WAKEUP) bit is set.
5. Multipurpose clock generator loss-of-clock (LOC). The LOC reset will assert if the clock monitor is enabled when entering stop or low power modes. This is cause by the internal slow IRC stopping in these low power modes. The slow IRC is used as a reference to the clock monitor.
6. MCG loss-of-lock (LOL) reset
7. Stop mode acknowledge error (SACKERR)
8. Software reset (SW)
9. Core Lockup reset (LOCKUP)
10. EzPort reset
11. MDM-AP system reset request

During the reset flow that occurs with the recovery from VLLSx low power modes, the WAKEUP bit in the SRS register is set and the following modules, LPTMR, RTC, CMP, TSI, PMC, SMC, are not reset. This detail is identified in the note before each register bit descriptions in the reference manual of the MCU. It is worded like this:

" NOTE This register is reset on Chip POR not VLLS and by reset types that trigger Chip POR not VLLS. It is unaffected by reset types that do not trigger Chip POR not VLLS. See the Reset section details for more information."

As noted in the LLWU reset the LPTMR registers are reset only on POR and LVD type resets. So this module can operate continuously through all power modes and all reset types except the POR and LVD resets. There are a number of other modules like this.

The LLWU reset occurs when recovering from VLLS0, VLLS1, VLLS2, or VLLS3. Using the SRS status register and mode controller control registers, you can identify whether the reset is from the LVD, POR, or an LLWU wake-up event. If it is an LLWU reset, the WAKEUP bit in the SRS register is set and the values in the mode controller control registers will identify which mode the MCU was recovering from.

How to manage all of the different reset types requires some additional consideration. Some modules do not get reset to default states with reset. Some or all of the RAM can be retained and will not require re-initialization by the startup code.

5.1.1 Hold on I/O and oscillator

When in VLLSx and LLS low power modes, and while exiting VLLSx low-power modes without the Reset pin, there is a hold on all of the I/O states and the system oscillator. The hold is released automatically when LLS is exited, but most I/O pins must be released manually when exiting from VLLSx modes.

GPIO: Most of the digital I/O pins on all Kinetis MCUs default to a high-impedance mode after a non-VLLSx reset recovery. However, upon a VLLSx (LLWU) reset recovery, the state of the pins is held. For instance, if a port pin was configured as an output driving high, it would continue to drive high until the hold was released. If the output port was not re-initialized before the hold was released, then the pin would temporarily become a high-impedance pin. This might glitch the output low for a short time until the software sets the pin to drive the correct output state.

OSC: If the oscillator is enabled to operate in VLLSx or LLS modes, then it continues to function when the MCU is in these modes and when exiting the VLLSx low-power modes. The only reason to enable the OSC in these low-power modes is to clock a module from the oscillator's ERCLK output.The registers that control the oscillator are reset when VLLSx modes are exited. The actual control signals are latched while in VLLSx modes. When the hold is released the oscillator control signals will be updated with the register values. If the oscillator is not configured correctly before releasing the hold it will stop running.

```
PMC_REGSC |= PMC_REGSC_ACKISO_MASK; //write to release hold on I/O
```

5.1.1 Releasing the hold: IF or WHEN

The hold on I/O and OSC is maintained in LLS mode and automatically released upon mode exit without software intervention.

VLLSx modes: The hold on I/O and OSC is maintained in VLLSx mode. During the reset recovery the hold is released automatically on the pins used for debug purposes and for the rest of the pins, when the software writes to the ACKISO bit in the PMC module.

The IF question:

There may be a time when you do not want to release the hold at all. You might want to check an internal condition, like the real time clock, and then go right back into one of the VLLSx low-power modes.

The WHEN Question:

As the MCU is waking from the VLLSx modes, and if you do not want to glitch your I/O or temporarily stop the oscillator, the code needs to sequence through the initialization process so that all of the I/O and the oscillator are re-initialized prior to releasing the hold. This, of course, is very application dependent..

1. Initialize the oscillator module and the MCG oscillator control bits as in this example

   ```
   // OSC_CR must enable external clock and enable in stop
   OSC_CR = OSC_CR_EREFSTEN_MASK | OSC_CR_ERCLKEN_MASK;
   ```
2. Initialize all digital I/O including GPIO, serial interfaces, comparators driving outputs, timers, and so on.
3. Release hold with write to ACKISO.

Because the required sequence is application dependent, none of the sample code or bare-bones projects have an example of this sequence.

5.2 Identifying reset types

There are methods to identify the new reset types shown in the previous section. The SRS registers, now residing in the Reset Control Module (RCM), along with the settings in the SMC control registers can identify which mode the MCU is recovering from.

The power-up states of the SRS register will indicate POR and LVD resets. If the POR and LVD bits are set in the SRS register, then you know that all MCU registers are set to their default state and the RAM and register file register are not initialized.

If there is another source of reset identified in the SRS registers, then you have some choices to make about how you handle the reset recovery. The reset flow after a recovery from VLLS0, VLLS1, VLLS2, or VLLS3 sets the WAKEUP bit in the SRS register. The WAKEUP bit indicates that the MCU is waking up via reset from these low-power modes. Using the fact that the WAKEUP bit is set you can interpret the control registers in the SMC, such as PMPROT, PMCTRL,STOPCTRL and on some MCUs the VLLSCTRL to identify which low-power modes you are exiting. These registers reflect the state that the MCU low power mode was entered and are not cleared by the LLWU reset.

5.3 Initialization of variables heap and stack space in C

Low-power modes LLS, LLS2, VLLS2, and VLLS3 have some or all of the RAM retained throughout the entry, the time in the low-power mode, and recovery. The Register File Modules, the system register file, and the RTC register file are maintained in all power-down modes including VLLS0 and VLLS1. The MCU also has built-in flash or EEPROM memory that can be used for non-volatile data storage.

The management of these memories can get creative. If the variable space, heap, and/or stack is retained, it need not be re-initialized during a reset flow.

5.3.1 RAM and register file retention

The entire RAM is not retained in all low-power modes; however, the register file contents are maintained. There is a handy table that describes the status of each of the modes in all of the low-power modes. It is the “Module operation in low power modes” table in the Power Management section of the reference manual. If you refer to this table, then you will note that RAM is retained in VLLS2, but no RAM is retained in VLLS1, VLLS0, or when the MCU is powered off.

5.3.2 Testing for low-power mode recovery

The C code below tests the SRS register bits for each reset source. Assuming that the wake-up bit was set in the SRS register, the SMC module registers are read to determine which low-power mode the MCU is recovering from. With this information, the recovering MCU can choose to bypass some of the tradition reset initialization.

NOTE

This is an example for rev 2.x Kinetis silicon, please consult the reference manual for the bit and register definitions for the MCU of choice.

```
if (RCM_SRS1 & RCM_SRS1_SW_MASK)
  printf("Software Reset\n");)
if (RCM_SRS1 & RCM_SRS1_LOCKUP_MASK)
  printf("Core Lockup Event Reset\n");
if (RCM_SRS1 & RCM_SRS1_JTAG_MASK)
  printf("JTAG Reset\n");
if (RCM_SRS0 & RCM_SRS0_POR_MASK)
  printf("Power-on Reset\n");
if (RCM_SRS0 & RCM_SRS0_PIN_MASK)
  printf("External Pin Reset\n");
if (RCM_SRS0 & RCM_SRS0_WDOG_MASK)
  printf("Watchdog(COP) Reset\n");
if (RCM_SRS0 & RCM_SRS0_LOC_MASK)
  printf("Loss of Clock Reset\n");
if (RCM_SRS0 & RCM_SRS0_LVD_MASK)
  printf("Low-voltage Detect Reset\n");
if (RCM_SRS0 & RCM_SRS0_WAKEUP_MASK)
{
  printf("[outSRS]Wakeup bit set from low power mode exit\n");
```

```
  printf("[outSRS]SMC_PMPROT   = %#02X \r\n", (SMC_PMPROT))  ;
  printf("[outSRS]SMC_PMCTRL   = %#02X \r\n", (SMC_PMCTRL))  ;
  printf("[outSRS]SMC_VLLSCTRL = %#02X \r\n", (SMC_VLLSCTRL));
  printf("[outSRS]SMC_PMSTAT   = %#02X \r\n", (SMC_PMSTAT))  ;

if ((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 3)
          printf("[outSRS] LLS exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 1))
          printf("[outSRS] VLLS1 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
    ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 2))
          printf("[outSRS] VLLS2 exit \n") ;
if (((SMC_PMCTRL & SMC_PMCTRL_STOPM_MASK)== 4) &&
   ((SMC_VLLSCTRL & SMC_VLLSCTRL_VLLSM_MASK)== 3))
          printf("[outSRS] VLLS3 exit \n") ;

}
```

5.4 Boot sequence

There are four basic boot sequences: normal boot from flash memory, boot from ROM, boot up in EZPORT mode, and reset with the debugger enabled. Consult the reference manual for your Kinetis device for the boot details.

Booting from flash has two options, normal boot and low-power boot. The main difference is how quickly the boot sequences happen and the current consumed during boot. To control this boot option, you must leave the LPBOOT bit. in the FOPT register set for normal boot, or configure the LPBOOT bits for low-power boot mode.

For Kinetis devices with EZPORT, EZPORT boot will occur if the EZPORT chip select input is held low during a POR or LVD reset. If this mode is not desired, you can opt out of the EZPORT boot mode by clearing the corresponding EZPORT disable bit in the FOPT flash register. With this bit clear, there will be no inadvertent EZPORT boot mode entries. If the EZPORT mode is needed at a later time, the MCU can be mass erased and the new setting for the FOPT register can be programmed.

The debug mode boot is not technically a separate boot mode, but is referenced here with reference to how it affects low power modes. If you reset the MCU with the debugger connected and active, some of the MCU low power modes act differently than they would in the normal operating mode of your application. The current reading of the MCU in STOP and VLPS modes is higher than in the applications mission mode (with the debugger disconnected) because the debugger module clocks are kept alive to allow for debugging through STOP and VLPS.

To ensure that your application is in mission mode, disconnect the debugger and power cycle the VDD to cause a POR reset. If the state of the RAM requires to be maintained a POR is not possible, so enter VLLS3 low power mode and recover and the debugger will be disabled.

6 Dynamic and Static Power Management

6.1 Power management through clock control

Clocking the circuitry of the MCU takes power. Careful control of the system clocking can make a large impact in the average current consumption of the application.

The faster the clock frequency the more power the circuit consumes. The MCU has many clocks and clock controls to enable the power management of the MCU through clock control. The MCU can function from internal or external clock sources and can use the internal reference clocks, FLL or PLLs to set the clock speed to the CPU, bus, the external bus, and the flash

memory and on some Kinetis MCUs the peripherals. The RTC module and RTC_OSC on some MCUs are independent clock domains from the core, platform and other peripherals. This means that even with the VDD or the MCU turned off, the RTC can retain autonomous operation to keep time or provide a wake-up event for the MCU.

Internal module power management control is possible. The peripheral modules have clock gate controls in the SIM that can disable an unused or idle module clock. This is an effective power management control on a per-module level.

6.2 Power management using low-power modes

Kinetis MCUs have up to eleven distinct power modes. There are four dynamic power modes and up to seven static power modes. To use these power modes for power management most effectively, it is important to understand the various trade-offs of each of the modes: what modules are available, how fast is the wake-up time, what memory is retained, what sections of the MCU are powered off, and what is the state of the I/O pins.

The resources needed to understand these trade-offs are identified throughout the MCU reference manuals.

7 Clock Operation in Low-power Modes

There are a number of clocks in the MCU. All can function in Run and Wait power modes. Some can be optionally turned on when operating in some of the low-power modes.

The clocks for the MCU are generated in one of the following type of clock modules. Refer to the reference manual of your Kinetis device for clock specifics.

1. Multipurpose Clock Generator Module (MCG)
2. Multipurpose Clock Generator Lite Module (MCG-Lite)
3. Internal Clock Source (ICS)

There are clocks generated in other modules including but not limited to:

1. System Oscillator (OSC)
2. Real Time Clock (RTC)
3. Power Management Controller (PMC)
4. USB crystal-less clock source (48MHz IRC)

These, along with the SIM, control the clock selection and distribution of these clocks in the various power modes. An example of how these modules fit together is shown in figure 9. For further study, and to understand the strict rules for clock mode transitions, please see the reference manuals.

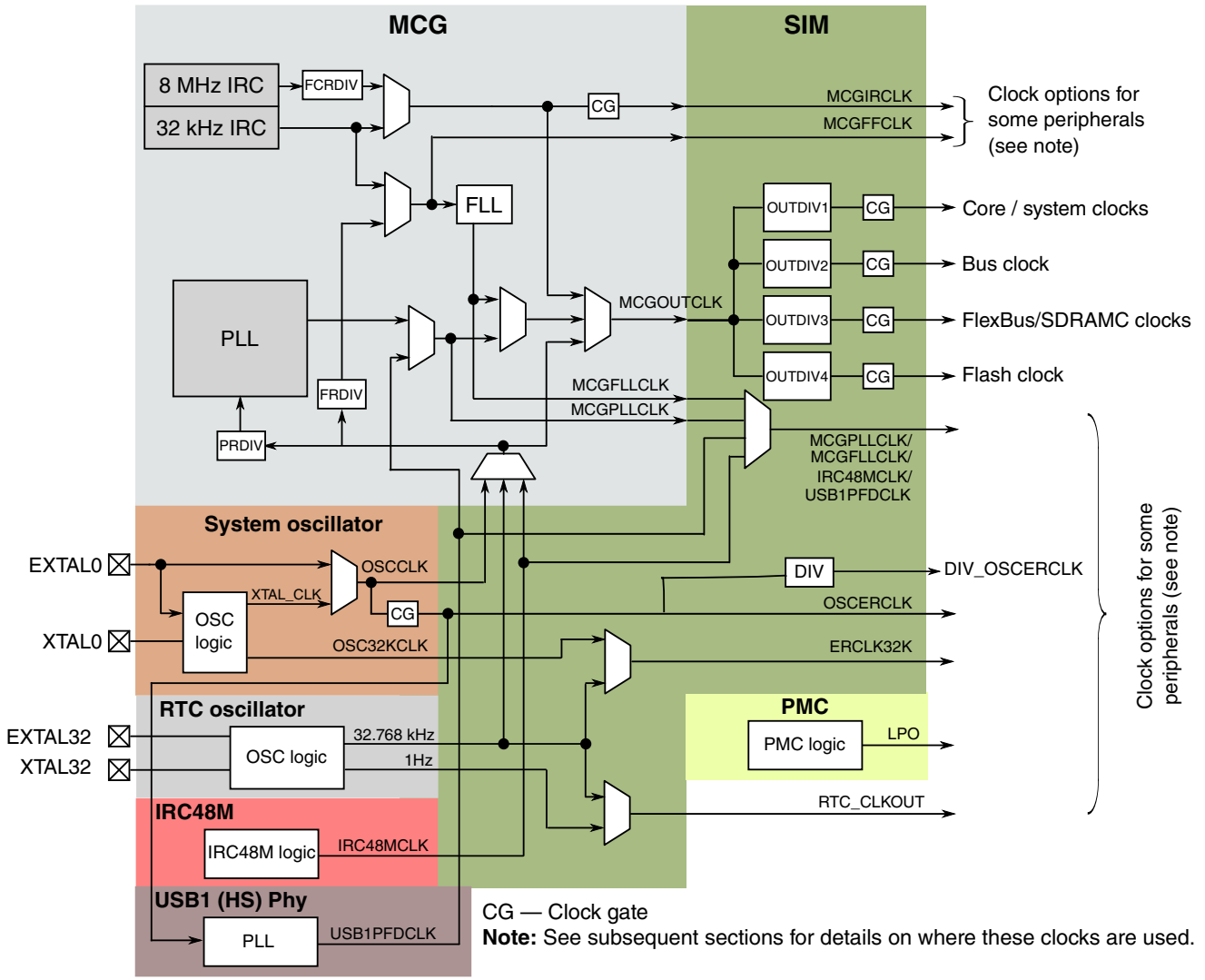


Figure 9. SOC clock diagram Example - Kinetis K66

7.1 Multipurpose Clock Generator (MCG)

The MCG is the main clock module. It works in conjunction with the oscillator modules to fulfill the clock needs of the MCU and peripheral modules. The MCG contains an FLL and up to two PLL clock generators. The FLL is less precise and is used to clock the MCU when there are no critical interface timing needs. The PLL is used to provide an accurate, low jitter clock to the CPU and peripherals when there is a need for this kind of accurate clock. The PLL is only capable of running with an external crystal or clock input source as the reference clock.

The MCG main system clock source can range from 0 to 150 MHz. There is also a separate PLL clock path available for the USB, I2S, and Ethernet modules. There is a separate FLL clock path available for other modules. MCGIRCLK is an internally generated clock that is available for use by the segment LCD, LPT, and TSI modules. MCGFFCLK is available for use by the Flex Timer (FTM).

A high speed IRC, called the IRC48, is available on some Kinetis MCUs. This can clock the MCU and also provides a means of clocking the USB in device mode without the need for an external crystal.

7.2 The MCG lite

The block diagram of MCG_Lite is provide. The first part to see this module was the Kinetis KL03

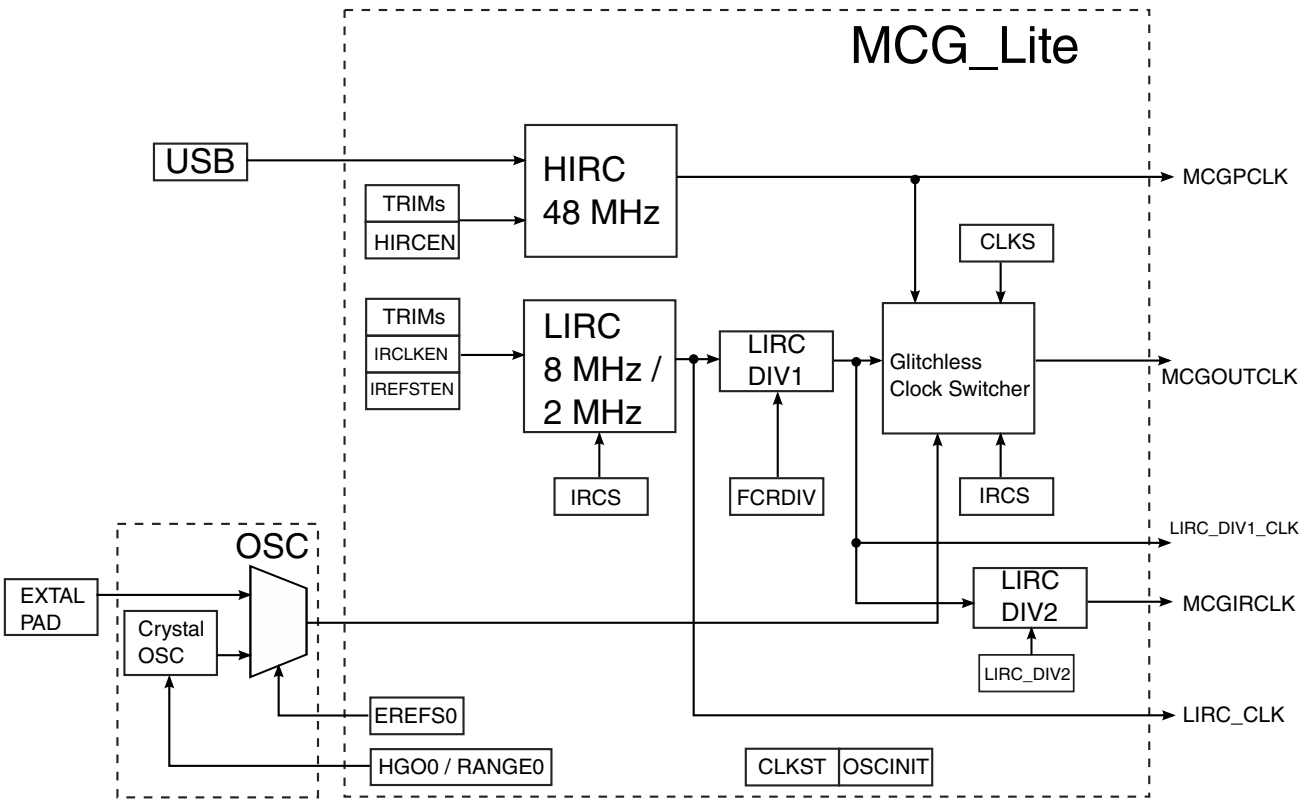


Figure 10. SOC clock diagram MCG LITE - Kinetis KL03

7.3 System Oscillator (OSC)

The OSCERCLK, from the OSC, is available for the FlexCAN, LPT, and ADC. The internal ERCLK32K is available for use by the segment LCD, LPT, and TSI.

7.4 RTC Oscillator (RTC)

The output of the OSC logic is sent to the Real Time Clock (RTC) and can be made available through ERCLK32K to other modules. The RTC oscillator can also be the clock source for the FLL (but not the PLL) and can be routed out to the rest of the system through MCGOUTCLK.

7.5 Power Management Controller (PMC) clocks

The PMC is the source of the low-power oscillator (LPO) that is made available as a clock source to modules. The LPO runs at a nominal frequency of 1 KHz +/– 30%. It is not very accurate over temperature and voltage but it can operate in all power modes. This clock can be the source of clock to a number of modules in the MCU including the LPTMR, the reset filter, the LLWU pin filters, and the LCD controller, so that they in turn can have a clock in the lowest of power modes. See the reference manual configuration sections for each module. The clock options available are detailed in tables.

7.6 Clock control in the SIM

The SIM provides control of the system clock dividers and various clock source multiplexors. The SIM also provides clock gate control of the individual modules in the device. Refer to the SIM and Clock distribution chapters of the individual device reference manual for details.

7.6.1 Clock gate control for peripheral modules in the SIM

There are up to seven clock gate control registers in the SIM. Most of the clock gate enable bits get cleared out of reset with a few exceptions. One is the bit that controls the FTFx—the clock enable for the flash memory.

You need to enable the clock gate to any module before you access it with a read or write, otherwise a hard fault reset will be generated. If you no longer need a module, or if it will be idle for an extended duration, the clock gate bit can be cleared. This will reduce power consumption from that module.

To minimize power consumption of the MCU in general, it is common practice to enable the clock gate only to the modules needed for the operation at hand. In this way, you can tune the power budget to the tasks required at that time.

If a module requires clocks for operation, such as a UART, disabling the clock gate will stop the module operation.

NOTE

The FTFx clock gate bit requires special handling. If entering LLS mode with FTFx cleared, the LLWU wake-up service routine and the interrupt vector must be located in internal RAM for the MCU to wake-up properly. If the flash is required upon exit from the LLWU service routine the FTFx clock gate must be re-enabled before exiting the ISR.

7.7 Clock operation in HSRun, Run and Wait power modes

High Speed Run allows maximum performance of chip. In this state, the MCU is able to operate at a faster frequency compared to normal run mode

The SIM has a number of clock controls. One of the most powerful is the SIMCLKDIV register, which allows clock divide control in Run mode, enabling dynamic frequency scaling to adjust the system power consumption according to the existing performance requirements. The maximum clock speed in Run and Wait modes is determined by the MCU specifications.

7.8 VLPR clock considerations

To enter VLPR from RUN, you have to ensure the VLPR clocking requirements are met. Since clock speeds are limited, so are the serial baud rates and timer timebases.

VLPW can only be entered from VLPR and share the exact same clock limitations. Even though VLPS can be entered directly from RUN mode, any peripherals that are being clocked in VLPS must not exceed the maximum frequencies as specified for VLPR. Refer to the individual device data sheet for device specific information.

The SIMCLKDIV system clock dividers must not be changed in VLPR mode. The desired system clock frequencies must be configured before entering VLPR. You cannot throttle the clock rates as you can in Run mode once you have entered VLPR.

The clock modes that can be used for VLPR and VLPW modes are BLPE, BLPI, or LIRC. BLPE modes uses an external clock source.

On some devices the LPBOOT bits in the FOPT register can be configured to directly enter VLPR mode after exiting from reset. The SIMCLKDIV dividers are configured to ensure the maximum VLPR system clock frequencies are not exceeded.

8 Power Mode Transitions

8.1 Entering low-power modes

Figure 11 shows the allowed power mode transitions. Any reset brings the chip back to the default run state as configured by the LPBOOT bits. In Run, Wait, and Stop modes, active power regulation is enabled. The VLPR and VLPW modes are limited in frequency, but offer a lower power operating mode than normal modes. The LLS and VLLSx modes are the lowest power Stop modes, and should be selected based on the amount of logic or memory that is required to be retained by the application.

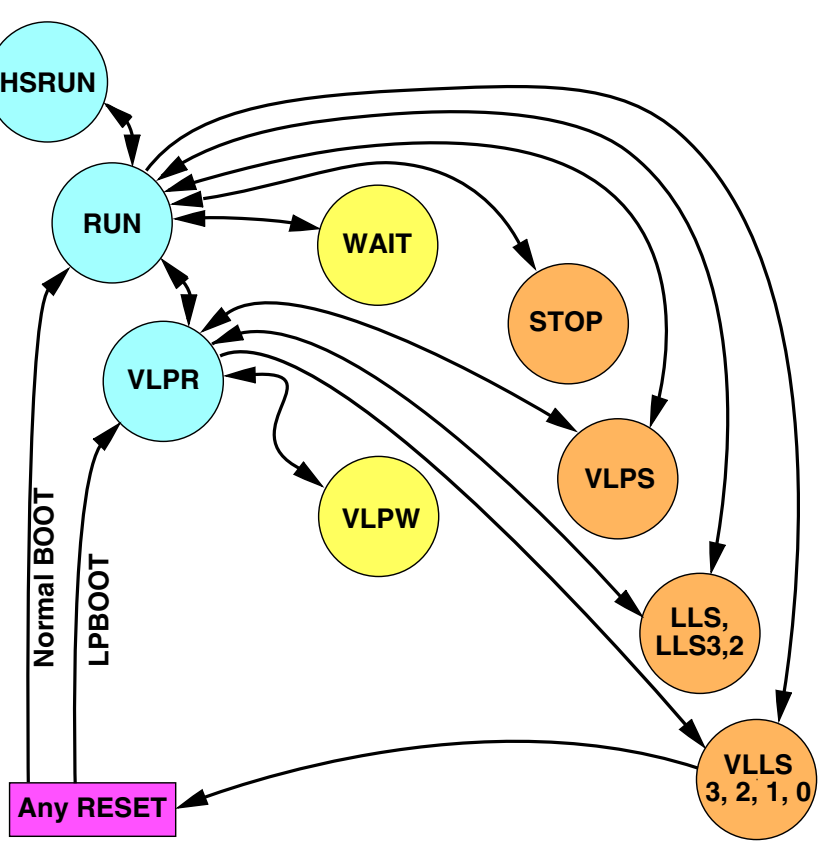


Figure 11. Power mode state transition diagram

8.1.1 Entering and exit conditions for each low power mode

Table 3. Power mode transition triggers

Transition #	From	To	Trigger conditions
1	RUN	WAIT	Sleep-now or sleep-on-exit modes entered with SLEEPDEEP clear, controlled in System Control Register in ARM core. See note.[1]
	WAIT	RUN	Interrupt or Reset
2	RUN	STOP	PMCTRL[RUNM]=00, PMCTRL[STOPM]=000 [2] Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	STOP	RUN	Interrupt or Reset
3	RUN	VLPR	The core, system, bus and flash clock frequencies and MCG,MCG_Lite clocking mode are restricted in this mode. See the Power Management chapter for the maximum allowable frequencies and MCG,MCG_Lite modes supported. Set PMPROT[AVLP]=1, PMCTRL[RUNM]=10. **NOTE:** In order to limit peak current, PMPROT[AVLP] and PMCTRL[RUNM] bits can be set on any Reset via Flash IFR settings, causing the SMC to transition the MCU from RUN->VLPR during the reset recovery sequence.
	VLPR	RUN	Set PMCTRL[RUNM]=00 or Interrupt with PMCTRL[LPWUI] =1 or Reset. **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
4	VLPR	VLPW	Sleep-now or sleep-on-exit modes entered with SLEEPDEEP clear, which is controlled in System Control Register in ARM core.
	VLPW	VLPR	Interrupt with PMCTRL[LPWUI]=0 **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
5	VLPW	RUN	Interrupt with PMCTRL[LPWUI]=1 or Reset **NOTE:** Not all devices have the PMCTRL[LPWUI] bit. For these devices the cases with LPWUI = 1 do not exist.
6	VLPR	VLPS	PMCTRL[STOPM]=000 [3] or 010, Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLPS	VLPR	Interrupt with PMCTRL[LPWUI]=0 **NOTE:** If VLPS was entered directly from RUN (transition #7), hardware forces exit back to RUN and does not allow a transition to VLPR. Not all devices have the PMCTRL[LPWUI] bit.

Table continues on the next page...

Table 3. Power mode transition triggers (continued)

Transition #	From	To	Trigger conditions
7	RUN	VLPS	PMPROT[AVLP]=1, PMCTRL[STOPM]=010 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLPS	RUN	Interrupt with PMCTRL[LPWUI]=1 or Interrupt with PMCTRL[LPWUI]=0 and VLPS mode was entered directly from RUN or Reset **NOTE:** Not all devices have the PMCTRL[LPWUI] bit.
8	RUN	VLLSx	PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), STOPE=1, WAITE=0 PMPROT[AVLLS]=1, PMCTRL[STOPM]=100,STOPCTRL[VLLSM]=x (VLLSx), Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	VLLSx	RUN	Wakeup from enabled LLWU input source or RESET pin
9	VLPR	VLLSx	PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), STOPE=1, WAITE=0 PMPROT[AVLLS]=1, PMCTRL[STOPM]=100, STOPCTRL[VLLSM]=x (VLLSx), Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
10	RUN	LLS	PMPROT[ALLS]=1, PMCTRL[STOPM]=011, STOPE=1, WAITE=0 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
11	LLS	RUN	Wakeup from enabled LLWU input source and LLS mode was entered directly from RUN or RESET pin.
12	VLPR	LLS	PMPROT[ALLS]=1, PMCTRL[STOPM]=011, STOPE=1, WAITE=0 Sleep-now or sleep-on-exit modes entered with SLEEPDEEP set, which is controlled in System Control Register in ARM core.
	LLS	VLPR	Implemented on some devices. Wakeup from enabled LLWU input source and LLS mode was entered directly from VLPR **NOTE:** If LLS was entered directly from RUN, hardware will not allow this transition and will force exit back to RUN
13	RUN	HSRUN	Set PMPROT[AHSRUN]=1, PMCTRL[RUNM]=11.
	HSRUN	RUN	Set PMCTRL[RUNM]=00 Reset

1. If debug is enabled, the core clock remains to support debug.

2. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=01 or 10, then only a Partial Stop mode is entered instead of STOP
3. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=00, then VLPS mode is entered instead of STOP. If PMCTRL[STOPM]=000 and STOPCTRL[PSTOPO]=01 or 10, then only a Partial Stop mode is entered instead of VLPS

9 Power Mode Entry Code

9.1 Power mode transition code

9.1.1 What is included and what is assumed

Example code for entering the different low-power modes is given in the next few sections. The short snippets demonstrate the simplicity of low-power mode entry.

The functions that do the work of entering the mode are included. Any required pre-mode initialization has been omitted, although there are notes indicating what is expected to be done by the time this code is executed.

9.1.2 Function Prototypes for the power mode entry drivers

```
/*****************************************************************/
// function prototypes
void sleep(void);
void deepsleep(void);
void enter_wait(void);
void enter_stop(void);
int enter_vlpr(char lpwui_value);
void exit_vlpr(void);
void enter_vlps(void);
void enter_lls(void);
void enter_lls2(void);
void enter_lls3(void);
void enter_vlls3(void);
void enter_vlls2(void);
void enter_vlls1(void);
void enter_vlls0(unsigned char PORPO_value);
/*****************************************************************/
```

9.1.3 Entering Sleep—create a function to enter Sleep

The ARM Cortex-M4 and M0+ cores uses the state of the SLEEPDEEP bit in the SCR to control which state the core platform enters when the WFI instruction is executed. If the SLEEPDEEP bit is clear, Sleep or Wait mode is entered.

```
/*****************************************************************/
/*
 * Configures the ARM system control register for WAIT(sleep)mode
 * and then executes the WFI instruction to enter the mode.
 *
 * Parameters:
 * none
 *
 */
```

```
void sleep (void)
{
/* Clear the SLEEPDEEP bit to make sure we go into WAIT (sleep)
 * mode instead of deep sleep.
 */
    SCB_SCR &=  ~SCB_SCR_SLEEPDEEP_MASK;
#ifdef CMSIS
    __wfi();
#else
/* WFI instruction will start entry into WAIT mode */
asm("WFI");
#endif
}
/*******************************************************************/
```

9.1.4 Entering Deep Sleep—create a function to enter Deep Sleep

The ARM Cortex-M4 and M0+ cores uses the state of the SLEEPDEEP bit in the SCR to control which state the core platform enters when the WFI instruction is executed. If the SLEEPDEEP bit is set Deep Sleep or core Stop mode is entered.

```
/*******************************************************************/
/*
 * Configures the ARM system control register for STOP
 * (deepsleep) mode and then executes the WFI instruction
 * to enter the mode.
 *
 * Parameters:
 * none
 *
 */

void deepsleep (void)
{
  /* Set the SLEEPDEEP bit to enable deep sleep mode (STOP) */
  SCB_SCR |= SCB_SCR_SLEEPDEEP_MASK;

#ifdef CMSIS
  __wfi();
#else
  /* WFI instruction will start entry into STOP mode */
  asm("WFI");
#endif
}
/*******************************************************************/
```

9.1.5 Entering Wait mode or VLPW

The MCU enters Wait or VLPW mode with the code below. The mode entered depends on the run mode in use at the time. If in run mode you enter Wait, if in VLPR mode you enter VLPW.

If the watchdog is enabled, then you must have a method to wake the device to service the watchdog, else the wake-up will be from a watchdog reset.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function if you are going to enter VLPW. For example, CME0 should also be written to a logic 0 before entering VLPR or VLPW power modes if the MCG is in BLPE mode.
- The wake-up events that are to be used to wake the MCU from Wait or VLPW must be set up before calling this function.

```
/*************************************************************/
/* WAIT mode entry routine. Puts the processor into Wait mode.
 * In this mode the core clock is disabled (no code executing),
 * but bus clocks are enabled (peripheral modules are
 * operational)
 *
 * Mode transitions:
 * RUN to WAIT
 * VLPR to VLPW
 *
 * This function can be used to enter normal wait mode or VLPW
 * mode. If you are executing in normal run mode when calling
 * this function, then you will enter normal wait mode.
 * If you are in VLPR mode when calling this function,
 * then you will enter VLPW mode instead.
 *
 * NOTE: Some modules include a programmable option to disable
 * them in wait mode. If those modules are programmed to disable
 * in wait mode, they will not be able to generate interrupts to
 * wake the core.
 *
 * WAIT mode is exited using any enabled interrupt or RESET,
 * so no exit_wait routine is needed.
 * For Kinetis K:
 * If in VLPW mode, the statue of the SMC_PMCTRL[LPWUI] bit
 * determines if the processor exits to VLPR (LPWUI cleared)
 * or normal run mode (LPWUI set). The enable_lpwui()
 * and disable_lpwui()functions can be used to set this bit
 * to the desired option prior to calling enter_wait().
 * For Kinetis L:
 * LPWUI does not exist.
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
 *
 * Parameters:
 * none
 */
void enter_wait(void)
{
    sleep();
}
/*************************************************************/
```

9.1.6 Entering normal Stop mode

The MCU enters normal Stop mode from run with the code below.

For Kinetis L the MCU enters VLPS instead of Stop mode if starting from VLPR mode.

If the watchdog is enabled, then you must have a method to wake the device to service the watchdog, else the wake-up will be from a watchdog Reset.

The MCU will always exit Stop mode into Run mode.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function.
- All modules that would not acknowledge a Stop mode entry must be handled. Refer to the reference manual for which modules need to handled.
- To enter Stop mode PMCTRL[RUNM]=00, PMCTRL[STOPM]=02.
- If the PSTOPO bit in the STOPCTRL register='b01 or 'b10, then only a Partial Stop mode is entered instead of STOP
- The wake-up events that are to be used to wake the MCU from Stop or VLPS must be set up before calling this function.

```
/*************************************************************/
/* STOP mode entry routine.
 * if in Run mode puts the processor into normal stop mode.
 * If in VLPR mode puts the processor into VLPS mode.
 * In this mode core, bus and peripheral clocks are disabled.
 *
 * Mode transitions:
 * RUN to STOP
 * VLPR to VLPS
 *
 * This function can be used to enter normal stop mode.
 * If you are executing in normal run mode when calling this
 * function and AVLP = 0, then you will enter normal stop mode.
 * If AVLP = 1 with previous write to PMPROT
 * then you will enter VLPS mode instead.
 *
 * STOP mode is exited using any enabled interrupt or RESET,
 * so no exit_stop routine is needed.
 *
 * Parameters:
 * none
 */
void enter_stop(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so, then this next write is not done since
     PMPROT is write once after RESET
     this write-once bit allows the MCU to enter the
     normal STOP mode.
     If AVLP is already a 1, VLPS mode is entered
     instead of normal STOP
     is SMC_PMPROT = 0  */

    /* Set the STOPM field to 0b000 for normal STOP mode
       For Kinetis L: if trying to enter Stop from VLPR user
       forced to VLPS low power mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_PMCTRL;
    deepsleep();
}
/*************************************************************/
```

9.1.7 Entering VLPR mode

The MCU enters VLPR mode with the code below. In Kinetis K, the value of LPWUI is passed as a parameter to this function. The state of LPWUI is static during VLPR, VLPW, and VLPS modes and should not be modified while in VLPR. For Kinetis L, the value you pass to this routine is a don't care.

Prerequisites:

- The frequency of the clock must be reduced. Some MCUs can operate with the system, bus, and FlexBus clocks at 4 MHz and the flash clock at 1 MHz.

  ```
  //core = /1 bus = /2 flexbus = /2 flash clk =/4
  SIM_CLKDIV1  =  (SIM_CLKDIV1_OUTDIV1(1) |
                   SIM_CLKDIV1_OUTDIV2(1) |
                   SIM_CLKDIV1_OUTDIV3(1) |
                   SIM_CLKDIV1_OUTDIV4(3));
  ```

- The clock monitor(s) in the MCG must be disabled before calling this function. For example, CME0 should also be written to a logic 0 before entering VLPR or VLPW power modes if the MCG is in BLPE mode.
- All modes that would not acknowledge a Stop mode entry must be disabled.

```
/*************************************************************/
/* VLPR mode entry routine.Puts the processor into Very Low Power
 * Run Mode. In this mode, all clocks are enabled,
 * but the core, bus, and peripheral clocks are limited
 * to 2 or 4 MHz or less.
 * The flash clock is limited to 1MHz or less.
 *
 * Mode transitions:
 * RUN to VLPR
 *
 * For Kinetis K:
 * While in VLPR, VLPW or VLPS the exit to VLPR is determined by
 * the value passed in from the calling program.
 * LPWUI is static during VLPR mode and
 * should not be written to while in VLPR mode.
 *
 * For Kinetis L:
 * LPWUI does not exist. the parameter pass is a don't care
 * Exits with an interrupt from VLPW will always be back to VLPR.
 * Exits from an interrupt from Wait will always be back to Run.
*
 * Parameters:
 * lpwui_value - The input determines what is written to the
 *               LPWUI bit in the PMCTRL register
 *               Clear LPWUI and interrupts keep you in VLPR
 *               Set LPWUI and interrupts return you to Run mode
 * Return value : PMSTAT value or error code
 *                PMSTAT = 000_0100 Current power mode is VLPR
 *                ERROR Code =  0x14 - already in VLPR mode
 *                           =  0x24 - REGONS never clears
 *                              indicating stop regulation
 */
int enter_vlpr(char lpwui_value)
{
  int i;
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK)== 4){
         return 0x14;
         }

  /* The PMPROT register may have been written by init code
     If so, then this next write is not done
     PMPROT is write once after RESET
     This write-once bit allows the MCU to enter the
     very low power modes: VLPR, VLPW, and VLPS   */
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  /* Set the (for MC1)LPLLSM or (for MC2)STOPM field
     to 0b010 for VLPS mode -
     and RUNM bits to 0b010 for VLPR mode
     Need to set state of LPWUI bit */
  lpwui_value &= 1;
  SMC_PMCTRL &= ~SMC_PMCTRL_RUNM_MASK;
  SMC_PMCTRL |= SMC_PMCTRL_RUNM(0x2) |
                (lpwui_value << SMC_PMCTRL_LPWUI_SHIFT);
  /* OPTIONAL Wait for VLPS regulator mode to be confirmed */
  for (i = 0 ; i < 10 ; i++)
    {     /* Check that the value of REGONS bit is not 0
             When it is a zero, you can stop checking */
      if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04){
       /* 0 Regulator is in stop regulation or in transition
             to/from it
          1 MCU is in Run regulation mode */
      }
      else  break;
    }
  if ((PMC_REGSC & PMC_REGSC_REGONS_MASK) ==0x04) {
       return 0x24;
     }
  /* SMC_PMSTAT register only exist in Mode Controller 2 */
```

```
  if ((SMC_PMSTAT & SMC_PMSTAT_PMSTAT_MASK) == 4) {
       return 0x04;
     }
}
/*****************************************************************/

/*  Use this statement in-line with your code to put
 *  the MCU into VLPR
 *  This assumes
 *  1)that the clocks are setup properly
 *  2)that you are entering VLPR from Run mode */

/*  add the next line if not already written. */
//  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;

  SMC_PMCTRL = SMC_PMCTRL_RUNM(0x2);
```

9.1.8 Entering VLPS mode

The MCU enters VLPS mode from Run or VLPR modes with the code below.

If the watchdog is enabled, then the you must have a method to wake the part to service the watchdog, else the wake-up will be from a watchdog reset.

Prerequisites:

- The clock monitor(s) in the MCG must be disabled before calling this function. For example, the CME0 bit should be written to a logic 0 before the MCG enters any Stop mode. Otherwise, a reset request may occur while in Stop mode.
- To enter VLPS mode, the ALVP bit in the PMPROT register must be set and the STOPM bit in the PMCTRL register must be 'b10.

```
/*****************************************************************/
/* VLPS mode entry routine. Puts the processor into VLPS mode
 * directly from run or VLPR modes.
 *
 * Mode transitions:
 * RUN  to VLPS
 * VLPR to VLPS
 *
 * Kinetis K:
 * when VLPS is entered directly from RUN mode,
 * exit to VLPR is disabled by hardware and the system will
 * always exit back to RUN.
 *
 * If however VLPS mode is entered from VLPR the state of
 * the LPWUI bit determines the state the MCU will return
 * to upon exit from VLPS.If LPWUI is 1 and an interrupt
 * occurs you will exit to normal run mode instead of VLPR.
 * If LPWUI is 0 and an interrupt occurs you will exit to VLPR.
 *
 * For Kinetis L:
 * when VLPS is entered from run an interrupt will exit to run.
 * When VLPS is entered from VLPR an interrupt will exit to VLPS
 * Parameters:
 * none
 */
/*****************************************************************/

void enter_vlps(void)
{
  volatile unsigned int dummyread;
  /*The PMPROT register may have already been written by init
     code. If so then this next write is not done since
     PMPROT is write once after RESET.
```

```
     This Write allows the MCU to enter the VLPR, VLPW,
     and VLPS modes. If AVLP is already written to 0
     Stop is entered instead of VLPS*/
  SMC_PMPROT = SMC_PMPROT_AVLP_MASK;
  /* Set the STOPM field to 0b010 for VLPS mode */
  SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
  SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x2);
  /*wait for write to complete to SMC before stopping core */
  dummyread = SMC_PMCTRL;
  /* Now execute the stop instruction to go into VLPS */
  deepsleep();
}
/*****************************************************************/
```

9.1.9 Entering LLS mode

The MCU enters LLS mode from Run or VLPR with the following code.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS
 * VLPR to LLS
 *
 * Wake-up from LLS mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b00 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(0);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.10 Entering LLS2 mode

The MCU enters LLS2 mode from Run or VLPR with the following code.

The mode that the MCU will exit from LLS2 is always Run mode.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS2 mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*************************************************************/
/* LLS mode entry routine. Puts the processor into LLS mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS2
 * VLPR to LLS2
 *
 * NOTE: LLS2 mode will always exit to Run mode even if you were
 * in VLPR mode before entering LLS2.
 *
 * Wake-up from LLS2 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls2(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b10 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(2);

   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*************************************************************/
```

9.1.11 Entering LLS3 mode

The MCU enters LLS3 mode from Run or VLPR with the following code.

Prerequisites:

- All modes that would not acknowledge a Stop mode entry must be disabled.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- To enter LLS3 mode, the ALLS bit in the PMPROT register must be set.
- If the ALLS and AVLP bits in the PMPROT register = 0 then a write to bits in the PMCTRL is ignored.
- All I/O is held at state while the MCU is in LLS3 mode and is released automatically upon wake-up.
- The wake-up events that are to be used to wake the MCU from LLS3 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* LLS3 mode entry routine. Puts the processor into LLS3 mode from
 * normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to LLS3
 * VLPR to LLS3
 *
 * Wake-up from LLS3 mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in LLS3 mode, so make
 * sure to set up the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_lls3(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow LLS power modes this write-once
      bit allows the MCU to enter the LLS low power mode*/
   SMC_PMPROT = SMC_PMPROT_ALLS_MASK;
   /* Set the (for MC1) LPLLSM or
     (for MC2)STOPM field to 0b011 for LLS3 mode
      Retains LPWUI and RUNM values */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x3) ;
    /* set LLSM = 0b11 in SMC_VLLSCTRL (for MC4) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_LLSM(3);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_PMCTRL;
   /* Now execute the stop instruction to go into LLS */
   deepsleep();
}
/*****************************************************************/
```

9.1.12 Entering VLLS3 mode

The MCU enters VLLS3 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS3 is always a reset.

Prerequisites:

- To enter VLLS3 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS3.
- The clock monitor(s) in the MCG must be disabled before calling this function.

- All I/O is held at state while the MCU is in VLLS3 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS3 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*************************************************************/
/* VLLS3 mode entry routine. Puts the processor into
 * VLLS3 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS3
 * VLPR to VLLS3
 *
 * NOTE: VLLSx modes will always exit to Run mode even if you were
 *       in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls3(void)
{
   volatile unsigned int dummyread;
    /* Write to PMPROT to allow VLLS3 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ; //(for MC1)
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ; //(for MC1)
    /* set VLLSM = 0b11 in SMC_VLLSCTRL (for MC2) */
     SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_CLLSCTRL;
    /* Now execute the stop instruction to go into VLLS3 */
    deepsleep();
}
/*************************************************************/
```

9.1.13 Entering VLLS2 mode

The MCU enters VLLS2 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS is always a reset.

For Kinetis L, MCU enters VLLS1 mode; VLLS2 mode is not supported on Kinetis L.

Prerequisites:

- To enter VLLS2 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS2.
- The clock monitor(s) in the MCG must be disabled before calling this function.

- All I/O is held at state while the MCU is in VLLS2 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS2 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*************************************************************/
/* VLLS2 mode entry routine. Puts the processor into
 * VLLS2 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS2
 * VLPR to VLLS2
 *
 * NOTE: VLLSx modes will always exit to Run mode even
 *       if you were in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls2(void)
{
    volatile unsigned int dummyread;
     /* Write to PMPROT to allow VLLS2 power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;

    /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
      or STOPM field to 0b100 for VLLSx (for MC2)
    - Retain state of LPWUI and RUNM */
     SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
     SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b10 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(2);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS2 */
   deepsleep();
}
/*************************************************************/
```

9.1.14 Entering VLLS1 mode

The MCU enters VLLS1 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS1 is always a reset.

Prerequisites:

- To enter VLLS1 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS1.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- All I/O is held at state while the MCU is in VLLS1 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS1 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake with the PTE1 pin.

```
/*****************************************************************/
/* VLLS1 mode entry routine. Puts the processor into
 * VLLS1 mode from normal Run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS1
 * VLPR to VLLS1
 *
 * NOTE:VLLSx modes will always exit to Run mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * none
 */
void enter_vlls1(void)
{
   volatile unsigned int dummyread;
   /* Write to PMPROT to allow all possible power modes */
   SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
   /* Set the VLLSM field to 0b100 for VLLSx(for MC1)
     or STOPM field to 0b100 for VLLSx (for MC2)
   - Retain state of LPWUI and RUNM */
   SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK ;
   SMC_PMCTRL |= SMC_PMCTRL_STOPM(0x4) ;
 //    MC_PMCTRL &= ~MC_PMCTRL_VLLSM_MASK ;
 //    MC_PMCTRL |= MC_PMCTRL_VLLSM(0x4) ;
   /* set VLLSM = 0b01 in SMC_VLLSCTRL (for MC2) */
   SMC_VLLSCTRL =  SMC_VLLSCTRL_VLLSM(1);
   /*wait for write to complete to SMC before stopping core */
   dummyread = SMC_VLLSCTRL;
   /* Now execute the stop instruction to go into VLLS1 */
   deepsleep();
}
/*****************************************************************/
```

9.1.15 Entering VLLS0 mode

The MCU enters VLLS0 mode from Run or VLPR with the code below.

The mode that the MCU will exit from VLLS0 is always a reset.

Prerequisites:

- To enter VLLS0 mode, the AVLLS bit in the PMPROT register must be set.
- If the AVLLS and AVLP bits in the PMPROT register = 0 then normal Stop is entered instead of VLLS1.
- The clock monitor(s) in the MCG must be disabled before calling this function.
- All I/O is held at state while the MCU is in VLLS0 mode and is released after reset when the code writes to the ACKISO bit.
- The wake-up events that are to be used to wake the MCU from VLLS0 must be set up in the LLWU and wake-up module before calling this function. See LLS mode entry and exit - bare metal code example section of this application note for an example of setting up the LLWU to wake the PTE1 pin.

```
/*****************************************************************/
/* VLLS0 mode entry routine. Puts the processor into
 * VLLS0 mode from normal run mode or VLPR.
 *
 * Mode transitions:
 * RUN to VLLS0
```

```
 * VLPR to VLLS0
 *
 * NOTE: VLLSx modes will always exit to RUN mode even if you were
 * in VLPR mode before entering VLLSx.
 *
 * Wake-up from VLLSx mode is controlled by the LLWU module. Most
 * modules cannot issue a wake-up interrupt in VLLSx mode, so make
 * sure to setup the desired wake-up sources in the LLWU before
 * calling this function.
 *
 * Parameters:
 * PORPO_value - 0 POR detect circuit is enabled in VLLS0
 *               1 POR detect circuit is disabled in VLLS0
 */
 /*****************************************************************/

void enter_vlls0(unsigned char PORPO_value )
{
    volatile unsigned int dummyread;
    /* Write to PMPROT to allow all possible power modes */
    SMC_PMPROT = SMC_PMPROT_AVLLS_MASK;
    /* Set the STOPM field to 0b100 for VLLS0 mode */
    SMC_PMCTRL &= ~SMC_PMCTRL_STOPM_MASK;
    SMC_PMCTRL |=  SMC_PMCTRL_STOPM(0x4);
    /* set VLLSM = 0b00 */
    SMC_VLLSCTRL =  (PORPO_value <<SMC_VLLSCTRL_PORPO_SHIFT)
                  | SMC_VLLSCTRL_VLLSM(3);
    /*wait for write to complete to SMC before stopping core */
    dummyread = SMC_VLLSCTRL;
    stop();
}
```

10 Power Mode Exit Transitions

10.1 More Notes about exiting low-power modes

The normal exit procedure is to enable and detect an event that initiates a mode change.

For Wait, Stop, VLPS, VLPR, and VLPW the wake-up event is typically an interrupt from a pin state change or module. The LLWU is not enabled and not used for these mode exits.

For LLS,LLS2,LLS3, VLLS3, VLLS2, VLLS1, and VLLS0 the wake-up events are limited to the enabled LLWU wake-up pins or modules, and the reset and NMI pins. The pin can be a wake-up source as long as the pin is "enabled as a digital input source" in the Pin Control Register. For MCU's without a LLWU, the module or pin wakeup is a wake-up source if included in the wake-up source table in the MCU reference manual and are set up as interrupt wake-up sources.

10.1.1 Pin Interrupt and LLWU Wake-up functionality integration

For LLWU wake-up input pins, it is recommended to also enable the pin interrupt functionality on the pin. In the event that the pin interrupt is not enabled, there is a window of time while the MCU transitions into the low leakage mode, that an edge transition might be missed. If the pin interrupt functionality was enabled and an edge occurred during this small transition time the low power entry would be aborted and the MCU would service the pin interrupt.

In Kinetis K devices all LLWU wake-up pins are also capable of being pin interrupt sources. In Kinetis L devices some LLWU wake-up pins do not have this dual functionality.

If the pin can also function as a pin interrupt to wake the MCU it is recommended to use both the interrupt and LLWU wakeup features. This is recommended since there is a very tiny window of time (~4ns) as the MCU transitions into the low leakage power mode, like LLSx or VLLSx, that an edge transition could be missed if the interrupt was not enabled for that pin. If the edge occurred during this small transition time the low power entry would be averted and the MCU would take the pin interrupt.

If the pin is configured as both a pin interrupt and an LLWU wake-up pin, the corresponding ISF flag bit in the port control register may be set and can be cleared in the LLWU. If the ISF flag is set and is not cleared in the LLWU ISR then the port ISR will be taken after the LLWU isr completes.

10.1.2 Reset as LLWU Wake-up

For all modes, the wake-up event could be a reset caused by any of the reset sources. These reset sources are listed in the reference manuals of the Kinetis MCU.

All references to the LPWUI bit are typically for Kinetis K MCUs. The Kinetis L series does not have the LPWUI bit.

10.1.3 Power Mode Transition Times

The interrupt latency of the M4 core based Kinetis devices is 12 cycles. The interrupt latency of the M0+ core based Kinetis devices is 15 cycles.

Table 4. Power Mode Transition Times

Transition number	Transition	Transition time
1	WAIT - RUN	Interrupt Latency
2	STOP - RUN	2us + Interrupt Latency
3	VLPW - VLPR	Interrupt Latency
4	VLPW - RUN	2 - 4 us + Interrupt Latency[1]
5	VLPS - VLPR	2 - 4 us + Interrupt Latency
6	VLPS - RUN	2 - 4 us + Interrupt Latency [2]
7	VLLSx - RUN	BOOT(LP) + 53 us to 115 us[3]
8	LLS - RUN	2 us + Interrupt Latency

1. The VLPW to RUN transition is only possible with Kinetis devices with LPWUI bit
2. The VLPS to RUN transition is only possible with Kinetis devices with LPWUI bit or Kinetis K22F MCUs
3. Exit from VLLSx modes is through the Wakeup Reset Flow

10.1.4 What can slow the exit from low power modes

- the MCG clock mode is not using the FLL. Using the MCG clock mode PEE causes the wakeup to start only after the external clock sources has started. This clock source is typically much slower than the PEE clock frequency.
- pin filtering is being used on the input trigger.
- The rise time of the input trigger is slow. (rising edge detected at VIH, falling edge at VIL).
- The MCU clock is slow or stopped.
- The wakeup source is the NMI pin and it stays low for a while. NMI is level sensitive and will stay in NMI ISR until NMI input returns high.

- The debugger is active during mode transitions. Interaction of the debug module can halt or stall processor.
- The external clock source is a crystal and it stops during low power mode. The crystal startup time is added to mode recovery time.

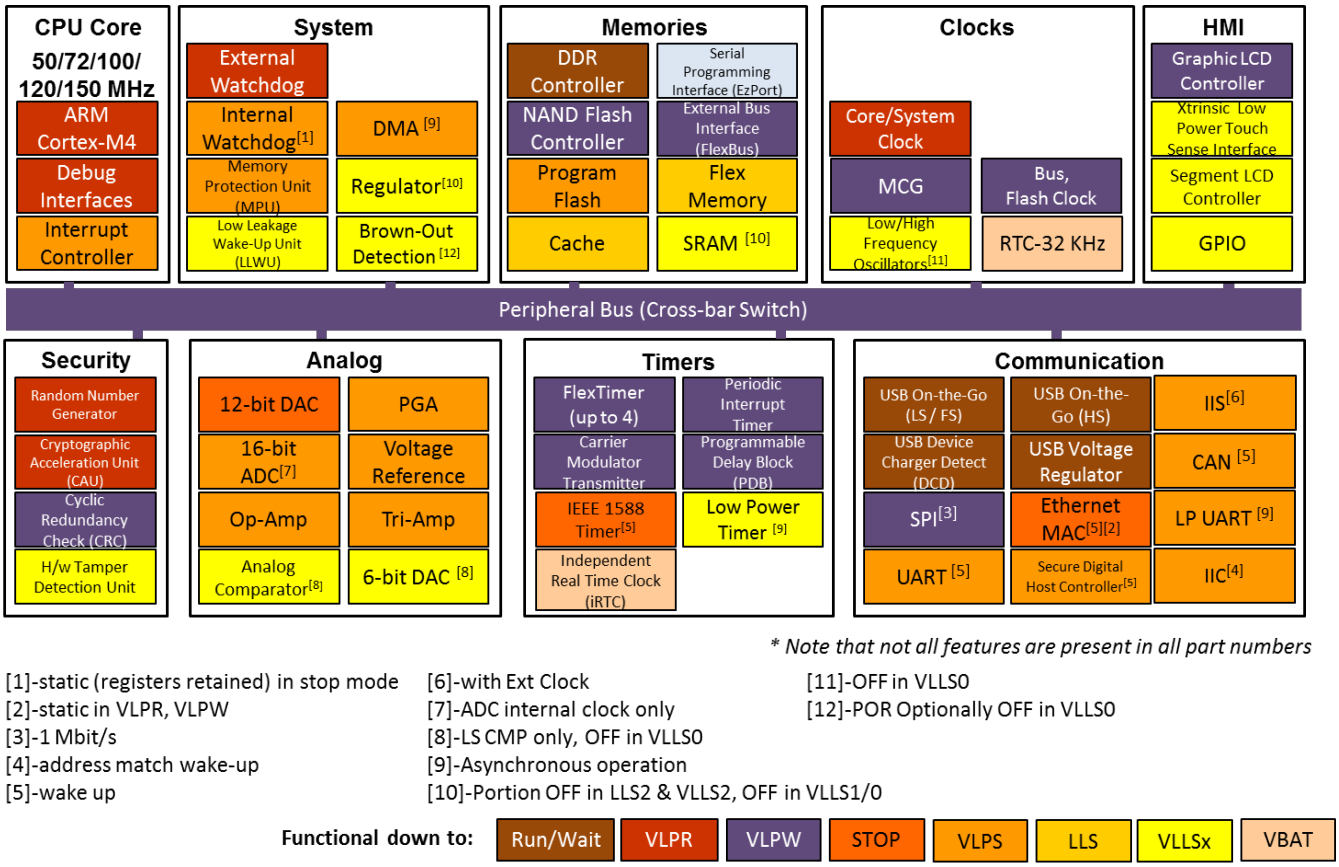
10.1.5 What can cause the MCU not to recognize an interrupt or LLWU input.

- The MCU is in RUN, WAIT, VLPR or VLPW and the input pulse is less than 1 and 1/2 bus clocks long. (i.e. A 250 ns input pulse with a 1 MHz bus won't work)
- pin filtering is being used on the input trigger and the input is not long enough. If LPO is used to clock the filter the pulse is less than 3 milliseconds
- The rise time of the input trigger is slow. (rising edge detected at VIH, falling edge at VIL).
- The MCU clock is slow or stopped.
- If you are in WAIT, VLPW, STOP, VLPS and global interrups are disabled while RESET and NMI are not used to wake up the MCU.
- If a pin is not a digital input, it is disabled or is an output the LLWU input function will not wake the MCU.

11 Modules in Power Modes

11.1 Module operation in low-power modes

The chart below illustrates the Kinetis K series module operation in the various low power modes.

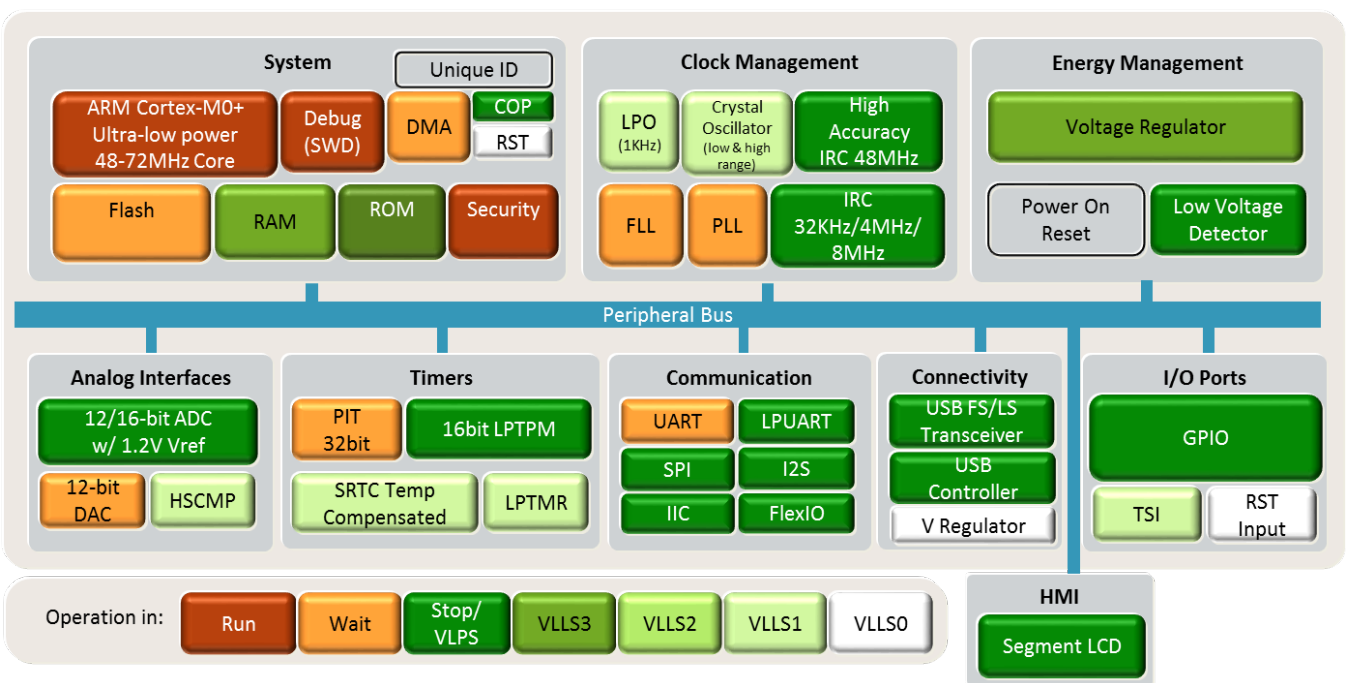


** Note that not all features are present in all part numbers*

[1]-static (registers retained) in stop mode
[2]-static in VLPR, VLPW
[3]-1 Mbit/s
[4]-address match wake-up
[5]-wake up
[6]-with Ext Clock
[7]-ADC internal clock only
[8]-LS CMP only, OFF in VLLS0
[9]-Asynchronous operation
[10]-Portion OFF in LLS2 & VLLS2, OFF in VLLS1/0
[11]-OFF in VLLS0
[12]-POR Optionally OFF in VLLS0

Functional down to: Run/Wait | VLPR | VLPW | STOP | VLPS | LLS | VLLSx | VBAT

Figure 12. Kinetis K-Modules Power Modes Chart



- FF—full functionality. In VLPR and VLPW, the system frequency is limited, but if a module does not have a limitation in its functionality, it is still listed as FF.
- Async operation = Kinetis L and later Kinetis K
- Static—module register states and associated memories are retained.

- Powered—memory is powered to retain contents.
- Low power—flash has a low-power state that retains configuration registers to support faster wake-up.
- OFF—modules are powered off; module is in reset state upon wake-up.
- Wake-up—modules can serve as a wake-up source for the chip.
- CPO - Compute Only Mode - Kinetis L and later Kinetis K
- PSTOPx - Partial Stop Mode 1 or 2 - Kinetis L and later Kinetis K
- IOPORT - Single Cycle IOPORT - Kinetis L and later Kinetis K

Table 5. Module operation in low-power modes

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
Core Platform modules						
CORE	static	FF	static	static	static	OFF
NVIC	static	FF	FF	static	static	OFF
FPU	static	FF	static	static	static	OFF
HDQR	static	FF	static	static	static	OFF
System modules						
Mode Controller	FF	FF	FF	FF	FF	FF
LLWU[1]	static	static	static	static	FF	FF[2]
Regulator	ON	low power	low power	low power	low power	low power in VLLS2/3, OFF in VLLS0/1
LVD	ON	disabled	disabled	disabled	disabled	disabled
Brown-out Detection	ON	ON	ON	ON	ON	ON in VLLS1/2/3, optionally disabled in VLLS0[3]
DMA	Async operation	FF Async operation in CPO	FF	Async operation	static	OFF
Watchdog	static FF in PSTOP2	FF static in CPO	FF	FF	static	OFF
EWM	static FF in PSTOP2	FF static in CPO	static	static	static	OFF
Clocks						
1kHz LPO	ON	ON	ON	ON	ON	ON in VLLS1/2/3, OFF in VLLS0
System oscillator (OSC)	OSCERCLK optional	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	limited to low range/low power	limited to low range/low power in VLLS1/2/3, OFF in VLLS0
MCG PLL	PLL optionally on but gated	static	static	static	static	OFF
MCG FLL	static—MCGIRCLK optional	static	static	static	static—no clock output	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
MCG Slow IRC	static - MCGIRCLK optional	static	static	static - MCGIRCLK optional	static—no clock output	OFF
MCG Fast IRC	MCGIRCLK optional	2 or 4 MHz IRC	2 or 4 MHz IRC	MCGIRCLK optional	static—no clock output	OFF
Core clock	OFF except in debug	2 or 4 MHz IRC	OFF	OFF	OFF	OFF
System clock	OFF except in debug	2 or 4 MHz max OFF in CPO	2 or 4 MHz IRC	OFF	OFF	OFF
Bus clock	OFF except in debug	1 MHz L series, 2 or 4 MHz IRC OFF in CPO	2 or 4 MHz IRC	OFF	OFF	OFF
Memory and memory interfaces						
Flash	powered	1 MHz max access—no pgm	low power	low power	OFF	OFF
Portion of SRAM_U[4]	low power	low power	low power	low power	low power	low power in VLLS3,2
Remaining SRAM_U and all of SRAM_L	low power	low power	low power	low power	low power	low power in VLLS3
Cache	low power	low power	low power	low power	low power	OFF
FlexMemory[5]	low power	low power[6]	low power	low power	low power	low power in VLLS3, OFF in VLLS2 and VLLS1
VBAT Register files[7]	powered	powered	powered	powered	powered	powered
System Register file	powered	powered	powered	powered	powered	powered
DDR controller	low power	low power	low power	low power	low power	OFF
SDRAM controller	low power	FF disabled in CPO	FF	low power	low power	OFF
NFC	static	FF	FF	static	static	OFF
FlexBus	static	FF disabled in CPO	FF	static	static	OFF
EzPort	disabled	disabled	disabled	disabled	disabled	disabled
Communication interfaces						
USB HS Phy	static	static	static	static	OFF	OFF
USB HS Controller	static, wakeup on resume	static, wakeup on resume	static, wakeup on resumeF	static	static	OFF
USB FS/LS	static, wakeup on resume	static, wakeup on resume	static, wakeup on resume	static	static	OFF
USB DCD	static	FF	FF	static	static	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
USB Voltage Regulator	optional	optional	optional	optional	optional	optional
Ethernet	wake-up	static	static	static	static	OFF
UART0, UART1	static, wakeup on edge	up to 250kbits/s static, wakeup on edge in CPO	up to 250kbits/s	static, wakeup on edge	static	OFF
LPUART Kinetis L & later Kinetis K	Async operation FF in PSTOP2	1-4 Mbps Async operation in CPO	1-4 Mbps	Async operation	static	OFF
FLEXIO Kinetis K2	Async operation FF in PSTOP2	1-4 Mbps Async operation in CPO	1-4 Mbps	Async operation	static	OFF
UART(other)	static, wake-up on edge FF in PSTOP2	125-250 kbit/s statis, wakeup on edge in CPO	125-250 kbit/s	static, wake-up on edge	static	OFF
SPI Kinetis L K2	static FF in PSTOP2	static, slave mode receive static in CPO	master mode 500 kbps, slave mode 250 kbps static, slave mode receive in CPO	static, slave mode receive	static	OFF
SPI	static FF in PSTOP2	1 Mbit/s (slave) 2 Mbit/s (master) static in CPO	1 Mbit/s (slave) 2 Mbit/s (master)	static	static	OFF
I^2C	static, address match wake-up	100 kbit/s	100 kbit/s	static, address match wake-up	static	OFF
CAN	wake-up FF in PSTOP2	250-500 kbit/s	250-500 kbit/s	wake-up	static	OFF
I^2C Kinetis L	50 kbps	static, address match wake-up FF in PSTOP2	50 kbps static, address match wake-up in CPO	static, address match wake-up	static	OFF
I^2C Kinetis K2	static, address match wake-up FF in PSTOP2	200 kbit/s static, address match wake-up in CPO	200 kbps	static, address match wake-up	static	OFF
I^2S	Async operation with external clock[8]	FF Async operation in CPO	FF	FF with external clock[9]	static	OFF
SDHC	wake-up	FF	FF	wake-up	static	OFF
Security						
CRC	static	FF	FF	static	static	OFF

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
RNG	static	FF	FF	static	static	OFF
DryIce [7]	FF	FF	FF	FF	FF	FF
MMCAU	static	FF	FF	static	static	OFF
Timers						
TPM Kinetis L K2	Async operation FF in PSTOP2	FF Async operation in CPO	FF	Async operation	static	OFF
FTM	static FF in PSTOP2	FF	FF	static	static	OFF
PIT	static FF in PSTOP2	FF static in CPO	FF	static	static	OFF
PDB	static FF in PSTOP2	FF static in CPO	FF	static	static	OFF
LPTMR Kinetis L K2	Async operation FF in PSTOP2	FF	FF	Async operation	Async operation	Async operation[10]
LPTMR	FF	FF	FF	FF	FF[11]	Async operation[10]
RTC Kinetis L K2	Async operation FF in PSTOP2	FF Async operation in CPO	FF	Async operation	Async operation	FF[12]
RTC - 32kHz OSC	FF	FF	FF	FF	FF[13]	FF
CMT	static	FF	FF	static	static	OFF
Analog						
16-bit ADC	ADC internal clock only	FF	FF	ADC internal clock only	static	OFF
CMP - Kinetis L[14]	HS or LS compare FF in PSTOP2	FF HS or LS compare in CPO	FF	HS or LS compare	LS compare	LS compare in VLLS1/3, OFF in VLLS0
CMP[15]	HS or LS compare	FF	FF	HS or LS compare	LS compare	LS compare in VLLS1/2/3, OFF in VLLS0
6-bit DAC	static FF in PSTOP2	FF static in CPO	FF	static	static	static, OFF in VLLS0
VREF	FF	FF	FF	FF	static	OFF
OPAMP	FF	FF	FF	FF	static	OFF
TRIAMP	FF	FF	FF	FF	static	OFF
PGA	FF	FF	FF	FF	static	OFF
12-bit DAC	static FF in PSTOP2	FF static in CPO	FF	static	static	static

Table continues on the next page...

Table 5. Module operation in low-power modes (continued)

Modules	Stop	VLPR	VLPW	VLPS	LLSx	VLLSx
Human-machine interfaces						
GPIO	static output, wake-up input FF in PSTOP2	FF IOPORT write only if in CPO	FF	static output, wake-up input	static output, pins latched, wake-up input	static output, pins latched, wake-up input
Segment LCD	FF	FF	FF	FF	FF[16]	FF[17]
Graphic LCD	static	FF	FF	static	static	OFF
TSI Kinetis L	Async operation FF in PSTOP2	FF Async operation in CPO	Async operation	Async operation	Async operation	Async operation
TSI	wake-up	FF	FF	wake-up	wake-up[17]	wake-up[11]

1. Using the LLWU module, the external pins available for this chip do not require the associated peripheral function to be enabled. It only requires the function controlling the pin (GPIO or peripheral) to be configured as a digital input to allow a transition to pass to the LLWU.
2. Since LPO clock source is disabled, filters will be bypassed during VLLS0.
3. [MC2]The VLLSCTRL[PORPO] bit in the SMC module controls this option.
4. A 4, 8, 16 32 KB portion of SRAM_U and 32KB SRAM L (with devices with PORPO) block is left powered on in low power mode VLLS2. There is no VLLS2 mode on Kinetis L.
5. FlexRAM is always powered in VLLS3. When the FlexRAM is configured for traditional RAM, optionally powered in VLLS2 mode. When the FlexRAM is configured for EEPROM, off in VLLS2 mode.
6. FlexRAM enabled as EEPROM is not writable in VLPR and writes are ignored. Read accesses to FlexRAM as EEPROM while in VLPR are allowed. There are no access restrictions for FlexRAM configured as traditional RAM.
7. These components remain powered in BAT power mode.
8. Use an externally generated bit clock or an externally generated audio master clock (including EXTAL).

   FF in PSTOP2
9. Use an externally generated bit clock or an externally generated audio master clock (including EXTAL).
10. LPO clock source is not available in VLLS0. Also, to use system OSC in VLLS0 it must be configured for bypass (external clock) operation. Pulse counting is available in all modes.
11. System OSC and LPO clock sources are not available in VLLS0
12. In VLLS0 the only clocking option is from RTC_CLKIN.
13. RTC_CLKOUT is not available.
14. CMP in stop or VLPS supports high speed or low speed external pin to pin or external pin to DAC compares. CMP in LLS or VLLSx supports only low speed external pin to pin or external pin to DAC compares. Windowed, sampled & filtered modes of operation are not available while in stop, VLPS, LLS, or VLLSx modes.
15. CMP in Stop or VLPS supports high speed or low speed external pin to pin or external pin to DAC compares. CMP in LLS or VLLSx supports only low speed external pin to pin or external pin to DAC compares. Windowed, sampled & filtered modes of operation are not available while in Stop, VLPS, LLS, or VLLSx modes.
16. End-of-frame wake-up not supported in LLS and VLLSx.
17. TSI wakeup from LLS and VLLSx modes is limited to a single selectable pin.

12 Using external memories and peripherals - Use case with DDR Memory Controller and DDR Low Power Modes

12.1 Controlling DDR memory and interface code example

The Kinetis K70/K61 is integrated with a dram (DDR) memory controller that has additional low power modes that will be referred to as DDR low power modes. The DDR controls can help achieve low power operation even when executing code from DDR memory. When operating with a DDR2 or LPDDR1 memory, the key to the lowest power system is managing the transitions between the MCU run modes and the memories and the MCUs low power modes.

This section of the application note explores some techniques to use that will allow low power operation of the Kinetis K70/K61 with the integrated DDR memory controller driving DDR2 or LPDDR memories. Sections below describe the DDR memory controls and the entry and exit steps to use to transition between DDR and MCU low power modes and run modes.

Use these descriptions, in conjunction with the application note demo code, to explore the controls and evaluate some different scenarios of the MCU and memory operation.

What about using the DDR in VLPR mode. Note that the DDR runs from the PLL. Therefore, before entering any MCU mode that would turn off the PLL such as VLPR, VLPS, STOP, LLS, VLLSx, the DDR must be put into a low power mode.

The same approach to low power mode entry can be applied to other external memory and peripheral types such as SDRAM, serial flash and sensors. Most of these device have low power modes that need to be set up prior to the MCU entering it's low power mode.

Note: The SDK low power mode entry functions have before and after callback functions that are ideal for this kind of code. The before setting up the LPDDR memory or peripheral before allowing the MCU to enter the low power mode and the after for the recovery operations. Please refer to the SDK API Reference Manual for details.

12.1.1 DDR power mode controls and Use

There are 5 low power modes for the External DDR Memories and Controller, LP modes 1 and 2 do not retain the contents of the memory.

Table 6. DDR Memory Low Power Mode Control with Register DDR_CR16_LPCTRL

Mode	Memory Type	Value	DDR Low Power Mode
1	DDR2 and LPDDR	0x10	DDR Memory Power-Down
2	DDR2 and LPDDR	0x08	DDR Memory Power-Down with Memory Clock Gating
3	DDR2 and LPDDR	0x04	DDR Memory Self-Refresh
4	DDR2 and LPDDR	0x02	DDR Memory Self-Refresh with Memory Clock Gating
5	LPDDR	0x01	DDR Memory Self-Refresh with Memory and Controller Clock Gating. This LP mode 5 cannot be entered Manually

12.1.2 Before MCU low power Mode Entry

The software flow would normally call a function to place the MCU into one of the low power modes like, Stop, VLPS, LLS or VLLS3. If using external memories that allow low power operation, the function sets the memory and memory controller state before entering the requested low power mode. When using external DDR memories in conjunction with the MCU low power modes, the entry into the low power modes is preceded with the sequence to put the DDR memory into it's low power mode, the DDR memory controller is disable and the MCU DDR pads are set to a lower power state.

The LLS low power mode entry code below includes the DDR controller and SIM_MCR register writes to put the DDR memory into mode 4 – DDR self refresh mode with clock gating. This is the lowest power mode for a DDR2 type memory.

```
    /* send command to memory to enter ddr low power mode 4 */
    DDR_CR16 = DDR_CR16_LPCTRL(0x02);     //bit 17 set of DDR_CR16
    SIM_MCR =   SIM_MCR_DDRDQSDIS_MASK | SIM_MCR_DDRCFG(1)
              | SIM_MCR_DDRS_MASK ;
    for (i = 0;i < 100;i++){
         if ( SIM_MCR & SIM_MCR_DDRS_MASK == 0x02)
         break;
         }
    SIM_SCGC3 &= ~SIM_SCGC3_DDR_MASK;
   // turned off the clock gate for the DDR Controller
```

12.1.3 After MCU low power Mode Exit

The MCU is now in LLS mode.

If a falling edge event occurs on PORTE1, the MCU will exit LLS and return to Run mode.

After exiting from LLS, code clears the wake-up event flag in the LLWU flag register

If interrupts and the LLWU interrupt are enabled, execution resumes with the LLWU interrupt service routine. Then code execution returns to the instruction following the WFI or STOP instruction.

If interrupts are disabled, the execution resumes with the instruction after the WFI, or STOP instruction.

At this point the DDR memory is disabled. The DDR cannot be accessed by any of the code below until the PLL restarts and the DDR memory exits self refresh. Be careful not to have any interrupts enabled that might attempt to access the DDR until the PLL is restarted and this code has completed.

```
  /* after exiting LLS (can be in LLWU interrupt service Routine)
  * clear the wake-up flag in the LLWU-write one to clear the flag
  */
  if (LLWU_F1 & LLWU_F1_WUF0_MASK) {
      LLWU_F1 |= LLWU_F1_WUF0_MASK;

 /* Enable DDR controller clock gate */
      SIM_SCGC3 |= SIM_SCGC3_DDR_MASK; /* Enable DDR controller clock gate */
     DDR_RCR = DDR_RCR_RST_MASK;     // reset DDR PHY

     DDR_RCR = 0;
     /* soft reset to DDR RCR, DDR_DQS analog circuit enabled,
        lpddr half strength, DDRPEN = 1 */
     SIM_MCR &= ~SIM_MCR_RCRRSTEN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRSREN_MASK;
     SIM_MCR &= ~SIM_MCR_DDRDQSDIS_MASK;
     DDR_CR50 |= DDR_CR50_CLKSTATUS_MASK;
     /* disable the DDR memory low power mode  */
     DDR_CR16 &= ~DDR_CR16_LPCTRL_MASK;  // all bits are zero
     DDR_CR16 |= DDR_CR16_QKREF_MASK;    // since set by lpddr_init code
```

```
    DDR_CR15 &= ~DDR_CR15_SREF_MASK;    // Disable self-refresh mode
    SIM_MCR |=  SIM_MCR_DDRPEN_MASK;
   /*wait for write to complete to before continuing */
   dummyread = SIM_MCR;
}
```

13 Power Measurement

13.1 Power measurement results and tips

13.1.1 Power measurement in real time

The typical way of measuring the current consumption of the application is with a digital multimeter. Measuring the current of a Kinetis MCU in these lowest power modes requires +/– 10 nA accuracy over a wide range.

A much better view of the real-time current consumption is available if you measure the voltage drop across a resistor or a 10 turn current loop in series with the VDD of the MCU.

Figure 13 and Figure 14 show the real-time current measurements across a 10 ohm resistor using an active differential probe and oscilloscope.

The software used to take these measurements was ported from a piece of demonstration code that repeatedly performed an ADC measurement of a single input, placing the results in a table. Every 256 samples, an FFT calculation using the Kinetis CMSIS DSP library was initiated. The ADC measurement and the FFT number crunching were done in either Run or VLPR mode. While not in Run mode, the MCU was put into VLPS mode.

Figure 13 shows the real time measurement results for two devices, K40 100 MHz MCU in Run mode and a K20 50 MHz MCU in VLPR mode.

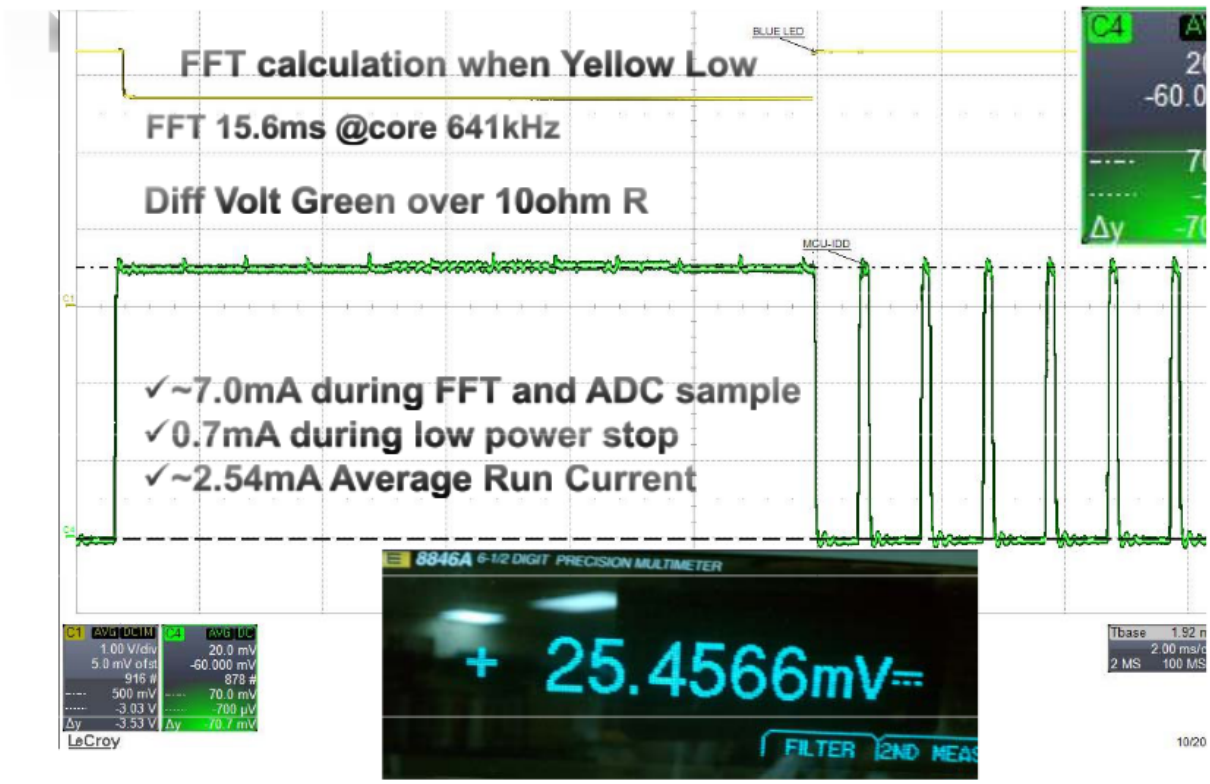


Figure 13. Real-time measurement of Run and VLPS modes

The measurements taken above were across a precision 10 ohm resistor using a differential probe. Divide the voltage measurement by 10 to get the value for current.

Run currents were approximately 7 mA and VLPS currents were approximately 700 µA. The average voltage taken with a precision digital multimeter was 2.54 mA.

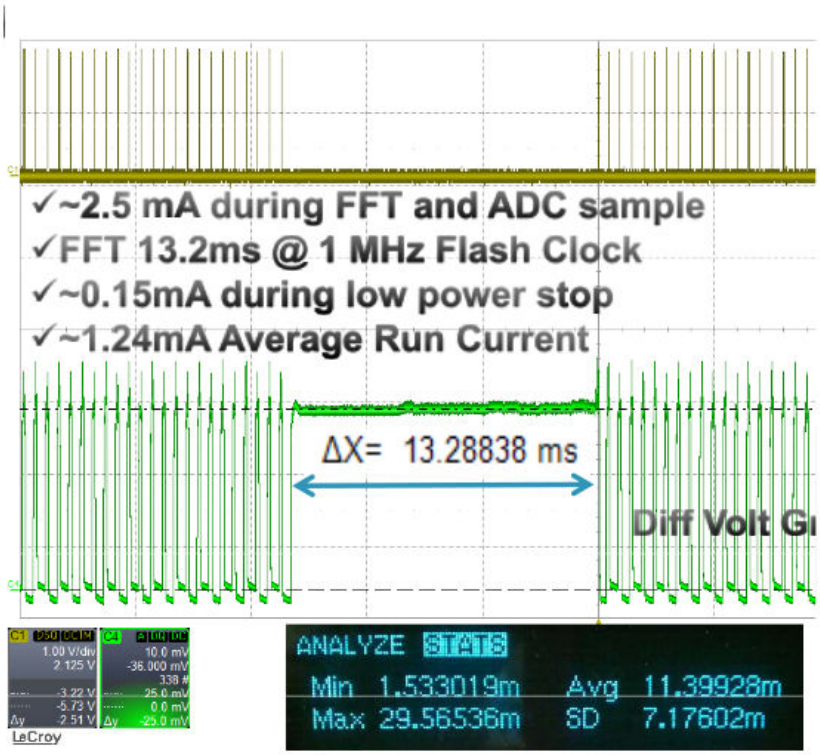


Figure 14. Real-time measurement of VLPR and VLPS modes

The measurement above is across a precision 10 ohm resistor using a differential probe. Divide the voltage measurement by 10 to get the value for current.

Measuring voltage across the same 10 ohm precision resistor using a precision digital multimeter with statistical calculations ability, the minimum, maximum, and average currents, as well as the standard deviation are superimposed at the bottom of the diagram.

Very Low Power Run currents ~ 2.5 mA and VLPS currents are ~ 150 µA. The average voltage taken with a precision digital multimeter (DMM) is 1.2 mA.

13.1.2 Tips for making low-power measurements on the bench

- **External.** The suggestions below address the most common issues encountered when trying to duplicate the data sheet current specs.
  a. **When using a digital multimeter (DMM), use "Manual Range Mode."** Using a DMM with an auto-ranging function enabled may cause LVD and POR resets. This is most common when you are exiting from one of the low-power modes like LLS or VLPS back to Run. The DMM has changed the range to a micro-amp or nano-amp range while the MCU is in the low-power mode and the sudden inrush of current requires the DMM to change range. The range change does not happen fast enough and the MCU starves and pulls the VDD level below the LVD or POR limits.
  b. **Disconnect the debugger and power cycle the MCU.** With the JTAG debugger for Kinetis is attached, the MCU may have the debugger module in the MCU active, clocking and consuming power. The external debugger hardware may also load the I/O of the JTAG port when attached. As a result, your low-power measurements will be higher than expected.

c. **Isolate the MCU VDDs.** If you want to measure the current draw of the MCU, then remove the other IC and component networks that are sourced by the voltage supply sourcing the MCU. For example, the early tower boards provided for Kinetis have a potentiometer connected between MCU_VDD and ground. A 5 K potentiometer across a 3.6 V supply pulls 720 µA. This is huge when considering that the MCU consumes around 1.5 µA in VLLS1.
d. **Measure VBAT or RTC VDD by itself.** The current supplying the RTC VBAT domain is typically much less than 1 µA while the RTC is running, keeping time. This very small IDD can be swamped by Run or VLPR currents.
e. **Match impedance of inputs.** If the impedance of high speed signals (fast edge transitions) are not well matched then the signals can "ring" and exceed the Vdd supply of the device. This can result in the signal providing current to the device through the input protection diodes. This is particularly true for high speed input clocks. This issue can result in negative IDD measurements while in the lowest power modes.
f. **Match voltage levels.** Although the MCU input pins are 5 V tolerant on some parts, when the MCU goes into the low-power modes, measurement of the current through MCU_VDD will be affected by any input higher than MCU_VDD. The higher input pin will back power the MCU through the input pin, resulting in negative IDD reading in low-power modes.
g. **Reduce pin loading of the MCU.** When the MCU sources current through the output pins, the power is being sourced through MCU_VDD. This is most evident when you output high frequency signals to an output pin as you might with the FlexBus clock and address/data pins.
h. Build code with debug hooks that do not need the Debugger or high speed serial interface that consume a lot of current.

- **Internal to the MCU.** Below is a list of the most common issues that can prevent you from getting to the lowest data sheet current specs.
  a. Watchdog is not disabled, causing resets. Disable or service the watchdog.
  b. The clock monitor is not disabled which may cause resets. Disable all of the clock monitors.
  c. A crystal oscillator is enabled in low power mode. The RTC oscillator, typically consumes <500 na of current.
  d. The CLKOUT signal is being output to a pin. Any pin that is constantly changing state will draw power.
  e. The requested low-power mode is not allowed with a corresponding bit in the PMPROT register. For example, if ALLS is not set in the PMPROT and the WFI instruction is executed, you will enter Stop mode and not LLS.
  f. The clock gate for a module is not enabled before it is read or written. This causes a reset before the MCU tries to enter the low-power mode.
  g. The clock gate for a module that must acknowledge the mode controller mode entry request is turned off prior to low-power mode entry. This will result in a Stop mode Acknowledge reset.
  h. Failure to un-comment out the call to the stop or sleep function after debugging is complete will keep you in the higher run current mode.
  i. If the MCU software is polling the RTC registers or VBAT Register File module, the current of the VBAT domain will increase 2–5 times over normal because of the constant accesses from the code. Therefore, to keep the lowest current, the RTC registers should not be polled.
  j. The frequency of wake-up events is too high, which means that the MCU spends more time in Run or VLPR mode than in a low-power mode. As shown above, the transition time from low-power mode to Run mode is quick. If the MCU only spends 9 ms in run and 1 ms in a low-power mode, the average current of the system will be considerably higher than if the MCU was running only 1 ms every 1 second.
  k. The MCU is running at a much higher frequency than is need to accomplish the work. Throttle the clock with the SIM CLK dividers or reduce the clock. Obviously, the higher the MCU frequency the higher the IDD of the MCU. Reduce the clock and lower the Run or Wait current. However there is some trade-off. If the current of Run mode can be tolerated, then getting work done as quickly as possible and going right back to low power mode is more advantageous than running a slower clock in Run mode.
  l. An input pin is floating without an internal or external pull device. This can result in 50-80 uA of current per pin. This include the JTAG or SWD pins. Disable the JTAG pins on portA or properly terminate the inputs.

14 LLWU pin and module Wakeup

14.1 LLWU hardware considerations

14.1.1 Pin wake-up events

The MCU wake-up logic is designed to use interrupts to wake the MCU from low power modes. It is recommended that interrupts be unmasked and the LLWU interrupt enabled. If global interrupts are masked or the LLWU interrupt is not enabled, the application may have unpredictable operation.

LLWU_Px are external pin inputs. Any digital input function multiplexed on the pin can be selected as the wake-up source.

You can choose whether a pin will cause a wake-up event with a rising edge, falling edge, or any edge. The LLWU controls the actions of each of the pins identified in the LLWU wake-up sources table. This is a select group of pins that, for the most part, does not change across all of the Kinetis family of devices. Kinetis L devices may only have a subset of the pins defined.

The pins listed in the table below can act as wake-up sources as long as the pin mux is actively selecting a digital input pin function such as a GPIO input or UART RX or RTS, or SPI slave select input in slave mode The wake-up event will not wake-up the MCU if the pin mux is not an LLWU wake-up pin, is a digital output or is selecting an analog function. When the pin is a digital input function, there is an asynchronous path from the pad through the Asynchronous Wakeup Interrupt Controller (AWIC) to the LLWU. The asynchronous path allows the wake-up event to pass to the LLWU without clocks and clock synchronizers. See the chip's signal multiplexing table for the digital input signal options and the LLWU wake-up pin list.

14.1.2 Integrating Pin Interrupt and LLWU Wake-up functionality

For LLWU wake-up input pins, it is recommended to enable the pin interrupt functionality on the LLWU wake-up pin. In the event that the pin interrupt is not enabled, there is a window of time while the MCU transitions into the low leakage mode, that an edge transition might be missed. If the pin interrupt functionality was enabled and an edge occurred during this small transition time the low power entry would be aborted and the MCU would service the pin interrupt.

In Kinetis K devices all LLWU wake-up pins are also capable of being pin interrupt sources. In Kinetis L devices some LLWU wake-up pins do not have this dual functionality.

14.1.3 Module wake-up events

LLWU_M0IF–M7IF are the LLWU internal peripheral interrupt flags.You can choose up to eight module wake-up sources in the LLWU. To complete the initialization of the wake-up source, the module must be enabled and set up to create an interrupt and the LLWU Module Flag enabled as a LLWU wake-up source. The interrupt from an enabled module will create a wake-up event.

After a wake-up with a peripheral, both the LLWU interrupt flag and the waking module's flag will be set in the NVIC pending interrupt register. If waking from LLS, the LLWU interrupt routine will be serviced first. If the LLWU service routine code does not clear the module flag and the module pending interrupt flag in the NVIC, then the module interrupt will be called at the completion of the LLWU interrupt service routine. If a module flag is cleared a read-back of the flag is recommended to make sure it is cleared before leaving the interrupt service routine.(see Serialization of operations section for more detail.

14.1.4 Filtered wake-up inputs

In Kinetis K there is a glitch filter function available for up to two external GPIO wake-up pins. In subsequent MCUs the glitch filter is based upon the internal LPO clock and therefore will act to reject any pulse shorter than three LPO cycles.

You can write the two filter control registers, LLWU_FILT1 and LLWU_FILT2, to select which of the 16 input pins would be sent to the two filters. It does not make much sense to enable a pin to one of the filters and also enable the same pin enable functions. Enabling both the pin and the filter enables bypasses the filter functionality since the MCU will always wake on the edge event. Do not choose the edge flag wake-up if the filter is being enabled for a pin. For example, to use LLWU_P7 for filter 1, write a 0b111 to the FILTSEL bits and enable the filter edge (FILTE) in the LLWU_FILT1 register. Then write 0b00 to the pin enable bits of the LLWU_PE2 register to disable any edges from causing a wake-up event.

14.1.5 Wake-up sources

NOTE

$\overline{\text{RESET}}$ is also a wake-up source, depending on the bit setting in the LLWU_CS register for MC1 or the LLWU_RST register for MC2. See the Reset filter setting in the Reset Control Module (RCM) for more information on how to configure the filtering on the Reset pin input.

Table 7. Example wake-up sources for LLWU inputs (see the reference manual Chip Configuration section for MCU-specific implementation)

Input	Wake-up source	Input	Wake-up source
LLWU_P0	PTE1/LLWU_P0 pin	LLWU_P12	PTD0/LLWU_P12 pin
LLWU_P1	PTE2/LLWU_P1 pin	LLWU_P13	PTD2/LLWU_P13 pin
LLWU_P2	PTE4/LLWU_P2 pin	LLWU_P14	PTD4/LLWU_P14 pin
LLWU_P3	PTA4/LLWU_P3 pin[1, 2]	LLWU_P15	PTD6/LLWU_P15 pin
LLWU_P4	PTA13/LLWU_P4 pin	LLWU_M0IF	LPTMR[3]
LLWU_P5	PTB0/LLWU_P5 pin	LLWU_M1IF	CMP0
LLWU_P6	PTC1/LLWU_P6 pin	LLWU_M2IF	CMP1
LLWU_P7	PTC3/LLWU_P7 pin	LLWU_M3IF	CMP2
LLWU_P8	PTC4/LLWU_P8 pin	LLWU_M4IF	TSI
LLWU_P9	PTC5/LLWU_P9 pin	LLWU_M5IF	RTC Alarm
LLWU_P10	PTC6/LLWU_P10 pin	LLWU_M6IF	DryIce (tamper detect)
LLWU_P11	PTC11/LLWU_P11 pin	LLWU_M7IF	RTC Seconds

1. For Kinetis Rev 1.x devices, a falling edge input that remains low on this pin when waking up the MCU from VLLSx modes with the EzPort enabled causes entry into EzPort mode during the reset sequence. A falling edge input that remains low on this pin when waking up the MCU from any non-VLLSx mode with the NMI function selected in its port control register asserts an NMI exception on low power mode recovery. The same occurs when recovering from VLLSx modes if EzPort is disabled; otherwise, EzPort mode is entered.
2. For MC2 devices the $\overline{\text{EZP\_CS}}$ signal is checked only on *Chip Reset not VLLS*, so a VLLS wake-up via a non-reset source does not cause EzPort mode entry. If NMI was enabled on entry to LLS/VLLS, asserting the NMI pin generates an NMI interrupt on exit from the low power mode. On Kinetis devices NMI can also be disabled via the FOPT[NMI_DIS] bit.
3. Requires the peripheral and the peripheral interrupt to be enabled. The LLWU's WUME bit enables the internal module flag as a wake-up input. After wake-up, the flags are cleared based on the peripheral clearing mechanism.

15 References and Revision History

15.1 References

The references listed below have additional information regarding power management for Kinetis. You can find a particular reference manual, data sheet, or errata report by choosing a device on the Kinetis (www.freescale.com/Kinetis) pages, and then selecting the Documentation tab. To find the application notes below, search for the document number.

15.1.1 Kinetis Microcontrollers (MCUs)

Kinetis MCUs consist of multiple hardware and software-compatible ARM® Cortex®-M0+ and -M4-based MCU series with exceptional low-power performance, scalability and feature integration. At the time of this application note writing there are 8 families

1. Kinetis K - Broad and Scalable and Compatibility from 50-180MHz, 32 KB to 2 M Flash,up to 256 KB RAM, floating point unit, security, analog, timers and serial interfaces
2. Kinetis L - Worlds most energy efficient microcontrollers, up to 48 MHz, 8KB-256KB Flash,up to 32 KB RAM,low power timer and smart peripherals
3. Kinetis E - Enhanced ESD/EMC performance, up to 48 MHz, 8 KB-128 KB Flash,up to 8K RAM,ADC, Flextimers,high current output.
4. Kinetis EA - Automotive grade qualification, up to 48 MHz, 8 KB-128 KB Flash,up to 8 K RAM,ADC, Flextimers,high current output.
5. Kinetis MINI - the worlds smallest ARM-based microcontrollers packaged in chip-scale packages.
6. Kinetis M - Metering and measurement applications, 50 MHz, 32 KB-128 KB Flash,up to 32 KB SRAM,high accuracy Sigma Delta analog front end, security and serial interfaces.
7. Kinetis V - Designed for motor control and digital power conversion, 75-200 MHz, 16 KB-2MB Flash,up to 256 KB SRAM, high speed analog& timing peripherals, math accelerators.
8. Kinetis W - Integrated with sub-1 GHz and 2.4 GHz RF transceivers, 48-50 MHz, 32 KB-512 KB Flash, up to 64 KB SRAM, optimized for embedded wireless solutions.

Realize that the system mode controllers on all of these MCUs are not the same. Even within one family grouping, such as the K20 family for instance, there are some significant improvements in the system mode controller over time. Please be diligent in reviewing each of the reference manuals and data sheets of the MCU you are designing with to make sure it has the features you need for your application.

Please see http://www.freescale.com/kinetis for the detailed documentation and updates.

15.1.2 Reference documents and links

There is a growing community of MCU applications designers that you can access through the following references.

- **MCU Reference Manuals.** The reference manuals contain MCU-specific implementation details in the Chip Configuration chapters and the module specific sections. Look for a detailed description of the Clocks, Resets and Power Management Features of each MCU.
- **MCU Data Sheet Specifications.** The data sheet includes all of the MCU specifications, including clock rates, low-power mode power consumption expectations.
- **Errata for MCUs.** Device errata identify what functionality and/or specification is not being met due to a problem with the MCU.
- **Kinestis SDK.** Software HAL and Driver Reference manuals in the KSDK downloaded docs folder.

- **AN4447, *Freescale MQX™ Low-Power Management*.** Freescale offers MQX, the full-featured and complimentary Real-Time Operating System (RTOS). Starting with version 3.8, MQX integrates a Low-Power Management (LPM) driver to take advantage of the low-power operating modes in MQX applications.
- **AN4470, *Using Low Power modes on the Kinetis Family***. Includes low-power mode entry demonstration code that can be used across Kinetis devices with mode controller 2.
- **Kinetis Solutions Advisor,**. Access and run the Kinetis Solutions Advisor to help select the MCU you need http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=SOLUTION_ADVISOR.
- **Kinetis Community.** Go here to access the Kinetis Community. https://community.freescale.com/community/kinetis
- **MBED Community** Go here to access the MBED community for Kinetis. http://developer.mbed.org/teams/Freescale/ or
- You can evaluate the low-power modes with the many Tower and Freedom kits available for Kinetis.

15.1.3 Revision History

The following table summarizes content changes since the previous release of this document.

Table 8. Revision History

Rev. No.	Date	Substantial Changes
1	Nov. 2011	Added details of newer Kinetis devices and new Low Power Modes including Kinetis L series MCUs.
2	Mar. 2015	Add details of newer Kinetis devices included K20, K61, K70, K22F, KVx, K65, K66, K24... Added Power Management Techniques section Remove power technology details Add Kinetis SDK example code for low power mode entry and exit Removed references to Coldfire+ MCUs Add updates to MCU clocks and diagrams including MCG Lite. Update document with new features and ideas throughout. Add section on what to do before entering Low Power modes. Use case includes DDR and LPDDR memories. Add tables on mode entry and exit Add reference to SDK API reference manual. Added mode entry notes and code for LLS2 and LLS3. Update power mode state diagram.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4503
Rev. 2
4/2015