AN12736

K32L2B Power Mode Switch Application

Rev. 0 — 02/2020 Application Note

Contents

1 Introduction

The K32L2B microcontroller family provides an ultra-low power feature for the power sensitive market. Several low power modes are implemented in this MCU family to meet this requirement. This application note show users details of each power modes and provides user case examples in the SDK power mode switch example demo. Tips are given for using each of the power modes.

The MCUXpresso SDK provides users with robust peripheral drivers, stacks, middleware,and example applications designed to simplify and accelerate application development on any NXP MCU. The MCUXpresso SDK is complimentary and includes full source code under a permissive open-source license for all hardware abstraction and peripheral driver software.

This application note focuses on the Power Management Controller (PMC), System Mode Controller (SMC), Multipurpose Clock Generator Lite version (MCG-Lite), and Low Leakage Wakeup Unit (LLWU).

2 Power modes on K32L2B MCU

2.1 Basic power modes in Cortex-M0+ core

The Arm® Cortex-M0+ uses the basic power modes of Arm Cortex-M architecture: Run, Sleep, and Deep Sleep. The Cortex-M0+ processor sleep modes reduce power consumption.

- Sleep mode: It stops the processor clock.
- Deep sleep mode: It stops the system clock and switches off the PLL and flash memory.

The system can generate spurious wakeup events, for example, a debug operation wakes up the processor. For this reason, software must be able to put the processor back into the sleep mode after such an event. A program might have an idle loop to put the processor back into sleep mode. To enter the low power modes (sleep/deep sleep), there are three instructions to inform the processor:

- **Wait For Interrupt (WFI)**: The WFI instruction causes immediate entry to the sleep mode. When the processor executes a WFI instruction, it stops executing instructions and enters the sleep mode.
- **Wait For Event (WFE)**: The WFE instruction causes entry to the sleep mode conditional on the value of a one-bit event register (set by SEV instruction). When the processor executes a WFE instruction, it checks the value of the event register as below:
  - 0 = The processor stops executing instructions and enters the sleep mode.
  - 1 = The processor sets the register to zero and continues executing instructions without entering the sleep mode.
- **Send the EVent (SEV)**: The SEV instuction causes an event to be signaled to all processors within a multiprocessor system. It also sets the local event register.

In the Cortex-M0+ core, the SCB register controls the behavior of entering low power modes after the WFI/WFE instruction.

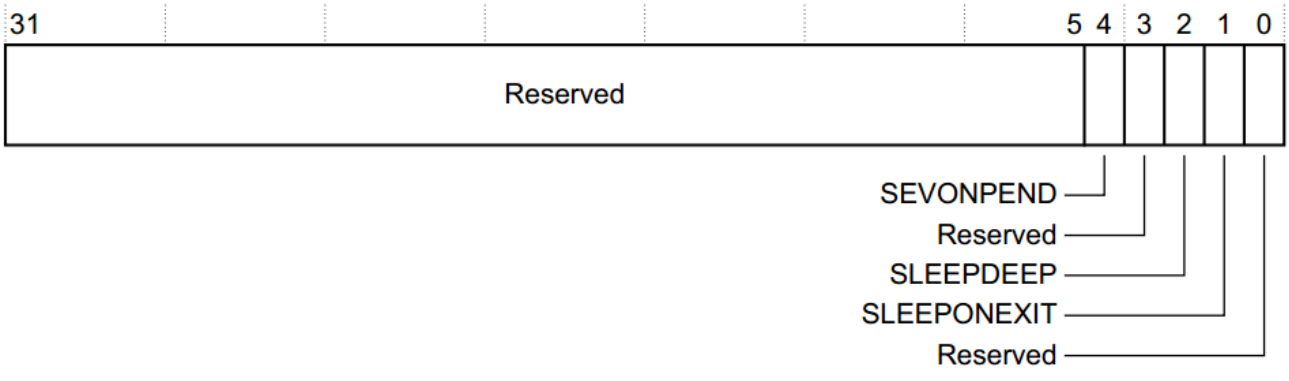


Figure 1. SCB register in Arm core

- `SCB[SLEEPDEEP]` bit controls whether the processor uses sleep mode or deep sleep mode as its low power mode:
  - 0 = Sleep
  - 1 = Deep sleep
- `SCB[SLEEPONEXIT]` bit indicates sleep-on-exit when returning from Handler mode (Interrupt Service Routine) to Thread mode (the `main()` function):
  - 0 = Do not sleep when returning to Thread mode, back to the `main()` function directly.
  - 1 = Enter the sleep or deep sleep again, on return from an ISR to Thread mode, never back to the `main()` function any more. Setting this bit to **1** enables an interrupt driven application to avoid returning to an empty main application.
- `SCB[SEVONPEND]` bit send Event on Pending bit:
  - 0 = Only enabled interrupts or events can wake up the processor and disabled interrupts are excluded.
  - 1 = Enabled events and all interrupts, including disabled interrupts, can wake up the processor. When an event or interrupt becomes pending, the event signal wakes up the processor from WFE. If the processor is not waiting for an event, the event is registered and affects the next WFE. The processor also wakes up on execution of an SEV instruction or an external event.

Here tells the **WFE** as a light weight version of **WFI** to pend the CPU's execution, but not restore and recover context, which saves more cycles to enter and exit the low power modes.

The WFE instruction causes entries to sleep mode conditional on the value of a one- bit event register. When the processor executes a WFE instruction, it checks the value of the event register:

- 0 = The processor stops executing instructions and enters the sleep mode.
- 1 = The processor sets the register to zero and continues executing instructions without entering the sleep mode.

If the event register is **0**, WFE suspends execution until one of the following events occurs:

- An exception, unless masked by the exception mask registers or the current priority level.
- An exception enters the Pending state, if SEVONPEND in the System Control Register is set.
- A Debug Entry request, if debug is enabled.
- An event signaled by a peripheral or another processor in a multiprocessor system using the SEV instruction.

2.2 Extend power modes in K32L2B

In the K32L2B, this core uses WFI instruction to invoke Sleep and Deep Sleep modes, but also extends power modes and their relationship, as presented in Table 1.

Table 1. Power modes on K32L2B MCU

Arm CM0+ power modes	K32L2B MCU power mode	Wakeup module	Reset or not?
RUN	RUN, VLPR	—	—
RUN	CPO	AWIC/NVIC	No
SLEEP	WAIT, VLPW	NVIC	No
DEEP SLEEP	STOP, VLPS	WIC	No
DEEP SLEEP	PSTOP1	AWIC	No
DEEP SLEEP	PSTOP2	AWIC/NVIC	No
DEEP SLEEP	LLS	LLWU	No
DEEP SLEEP	VLLSx (x=0/1/3)	LLWU	Yes

NVIC means any interrupt source can wake up an MCU from WAIT/VLPW mode. AWIC means only the AWIC wake-up source in the reference manual can wake up the MCU from STOP/VLPS mode. LLWU means only the LLWU wake-up source in the reference manual can wake up the MCU from LLS/VLLSx modes. To wake up from VLLSx mode, go through a reset flow and call LLWU reset. For Compute Operation mode (CPO), Arm core is in the run mode. Any asynchronous interrupt and Arm core synchronous interrupt can wake up the MCU to the run mode. Table 2 shows the detailed descriptions about each power mode.

Table 2. Power mode description

Mode	Description
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.
WAIT	The core clock is gated off. The system clock continues to operate. Bus clocks, if enabled, continue to operate. Run regulation is maintained.
STOP	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid.
VLPR	The core, system, bus, and flash clock maximum frequencies are restricted in this mode. See the **Power Management** chapter in *KL43 Sub-Family Reference Manual* (document KL43P64M48SF6RM) for details about the maximum allowable frequencies.
VLPW	The core clock is gated off. The system, bus, and flash clocks continue to operate, although their maximum frequency is restricted. See the **Power Management** chapter in *KL43 Sub-Family Reference Manual* (document KL43P64M48SF6RM) for details about the maximum allowable frequencies.
VLPS	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid.
LLS	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The MCU is placed in a low leakage mode by reducing the voltage to internal logic. All system RAM contents, internal logic and I/O states are retained.
VLLS3	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The MCU is placed in a low leakage mode by powering down the internal logic. All system RAM contents are retained and I/O states are held. Internal logic states are not retained.

Table continues on the next page...

Table 2. Power mode description (continued)

Mode	Description
VLLS1	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The MCU is placed in a low leakage mode by powering down the internal logic and all system RAM. I/O states are held. Internal logic states are not retained.
VLLS0	The core clock is gated off. System clocks to other masters and bus clocks are gated off after all stop acknowledge signals from supporting peripherals are valid. The MCU is placed in a low leakage mode by powering down the internal logic and all system RAM. I/O states are held. Internal logic states are not retained. The 1 kHz LPO clock is disabled and the power on reset (POR) circuit can be optionally enabled using `STOPCTRL[PORPO]`.

For K32L2B family devices, the NMI pin can wake up all power modes, while the reset pin resets MCU power mode into default RUN mode if the reset pin is not filtered by the bus clock.

Figure 2 shows the power mode state transitions available on the chip. Any reset always brings the MCU back to the normal RUN state.

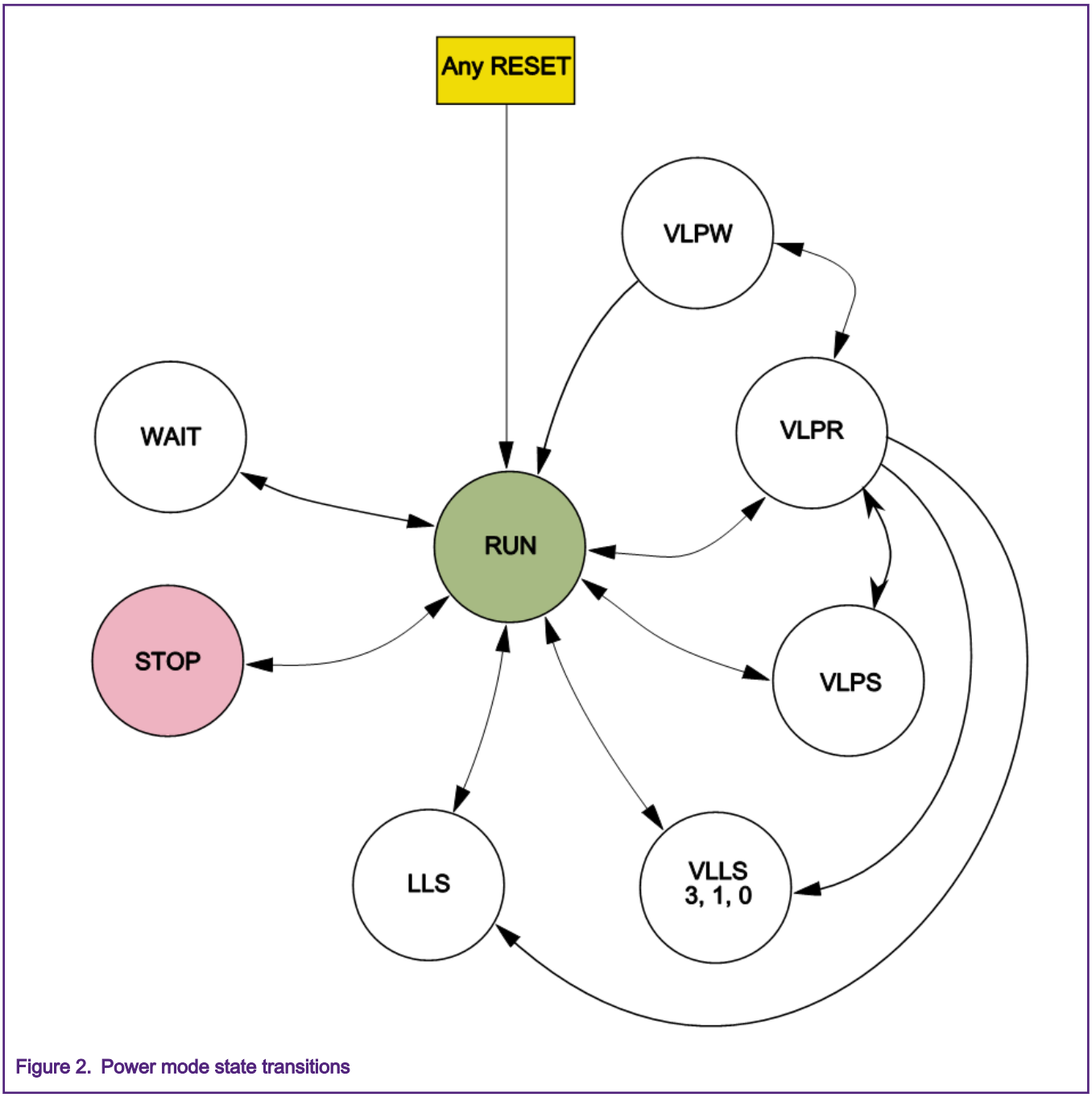


Figure 2. Power mode state transitions

To enter the target power mode from RUN/VLPR mode:

- Disable the Low Voltage Detection (LVD) functions in Power Management Controller (PMC) module, to ignore the warning when the LDO reduces the power supply in some ultra low power modes.
- Disable the unnecessary modules/pins to save power in target low power mode, and set up the clock wakeup source module to trigger the low power exit event.
- Set up the clock source for target power mode, so that the surviving module still be working with an available clock source.
- Unlock the indicated the power modes in **SMC** -> **PMPROT** (Power Mode Protection) reigister, so that the target power mode can be entered.
- Set up the target mode in **SMC** -> **PMCTRL** (Power Mode Control) register and the **SMC** -> **STOPCTRL** (Stop Control).

- Call the WFI or WFE to invoke into the low power mode.

To exit from the low power mode:

- Wait for the pre-setup wakeup source to be triggered by wakeup event.
- For the VLLSx modes, LLWU is specially desigend as a wakeup module to collect all available wakeup source.

For some wake-up routine from reset, it is also necessary to clear the **PMC -> REGSC[ACKISO]** bit to unlock the port pins, which is locked and kept stable in some ultra-low power modes.

About the wake-up source, Table 3 shows the surviving modules in the ultra low power modes (LLS and VLLSx).

Table 3. Surviving modules in ultra low power modes

Modules	VLPR	VLPW	Stop	VLPS	LLS	VLLSx
Core modules						
NVIC	FF	FF	static	static	static	OFF
System modules						
Mode Controller	FF	FF	FF	FF	FF	FF
LLWU	static	static	static	static	FF	FF
Regulator	low power	low power	ON	low power	low power	low power in VLLS3, OFF in VLLS0/1
Brown-out Detection	ON	ON	ON	ON	ON	ON in VLLS1/3, optionally disabled in VLLS0
Clocks						
1 kHz LPO	ON	ON	ON	ON	ON	ON in VLLS1/3, OFF in VLLS0
System oscillator (OSC)	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	OSCERCLK optional	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal	OSCERCLK max of 16 MHz crystal in VLLS1/3, OFF in VLLS0
Memory and memory interfacesSystem modules						
SRAM_U and SRAM_L	low power	low power	low power	low power	low power	low power in VLLS3, OFF in VLLS0/1
System register file	powered	powered	powered	powered	powered	powered
Timers						
LPTMR	FF	FF	Async operation	Async operation	Async operation	Async operation

Table continues on the next page...

Table 3. Surviving modules in ultra low power modes (continued)

Modules	VLPR	VLPW	Stop	VLPS	LLS	VLLSx
			FF in PSTOP2			
RTC	FF Async operation in CPO	FF	Async operation FF in PSTOP2	Async operation	Async operation	Async operation
Human-machine interfaces						
RTC	FF Async operation in CPO	FF	Async operation FF in PSTOP2	Async operation	Async operation	Async operation OFF in VLLS0

3 Application - measuring the current in various power modes

In the document, an application software is designed for measuring the current of K32L2B MCU working in various power modes. The FRDM-K32L2B board is used as main hardware platform. The two buttons on the board are used to switch the target power mode selection on SLCD screen.

3.1 Board settings

FRDM-K32L2B board has the measuring socket but need a little additional hardware work to make it available in application. In the schematic, J20 is the expected measurement socket, as shown in Figure 3.

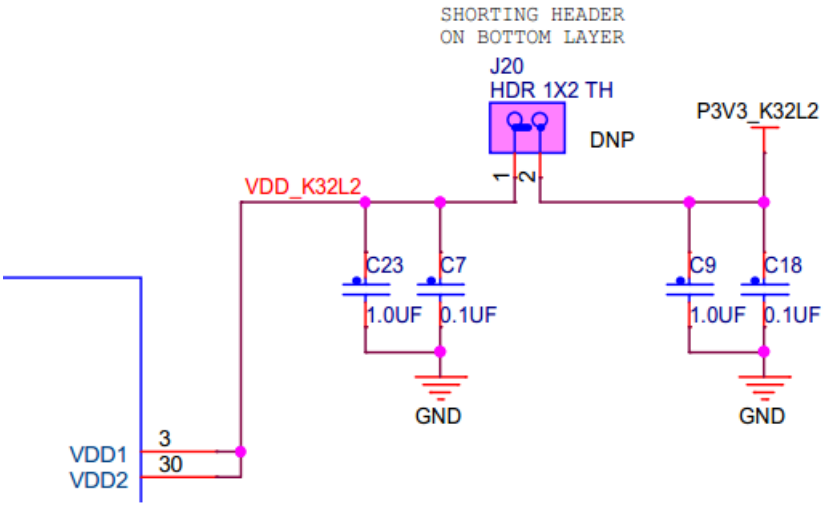
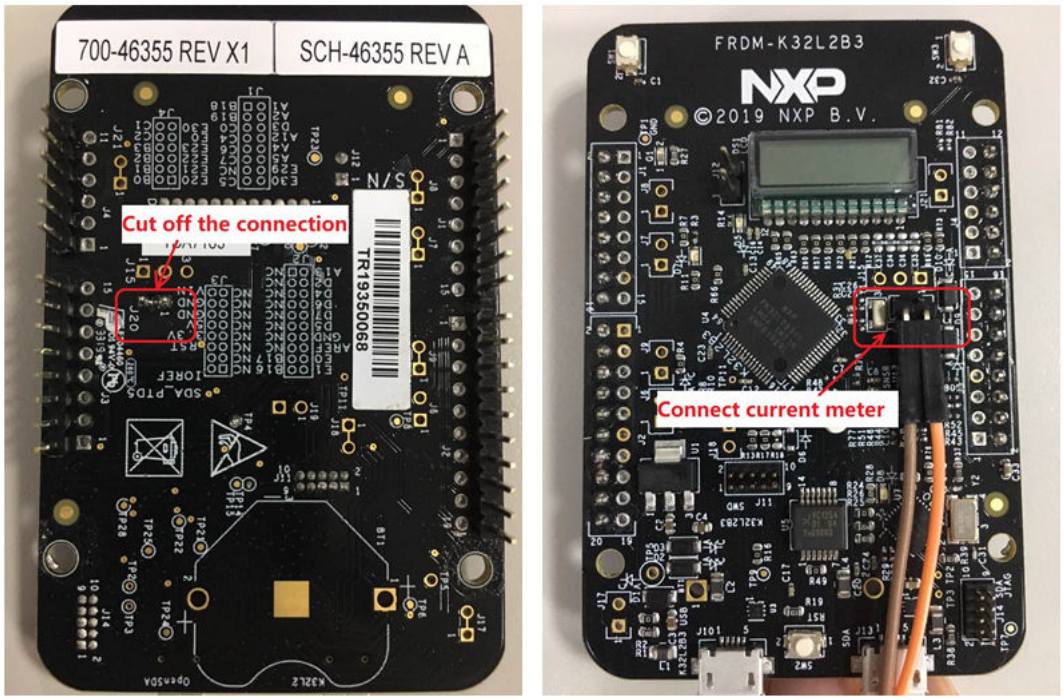


Figure 3. Schematic of measuring socket on FRDM-K32L2B board

However, J20 is shorted on the board by default. To measure the current going through into the VDD, we need to cut off this connection and put the multimeter (in current measurement mode) into the series connection, as shown in Figure 4.



A: Back side of J20

B: Front side of J20

Figure 4. FRDM-K32L2B measuring socket

To enable the SLCD displaying during the low power modes, use the on-board buttons (**SW1** and **SW3**) and SLCD screen to switch the power mode and show the status, as shown in Figure 5.

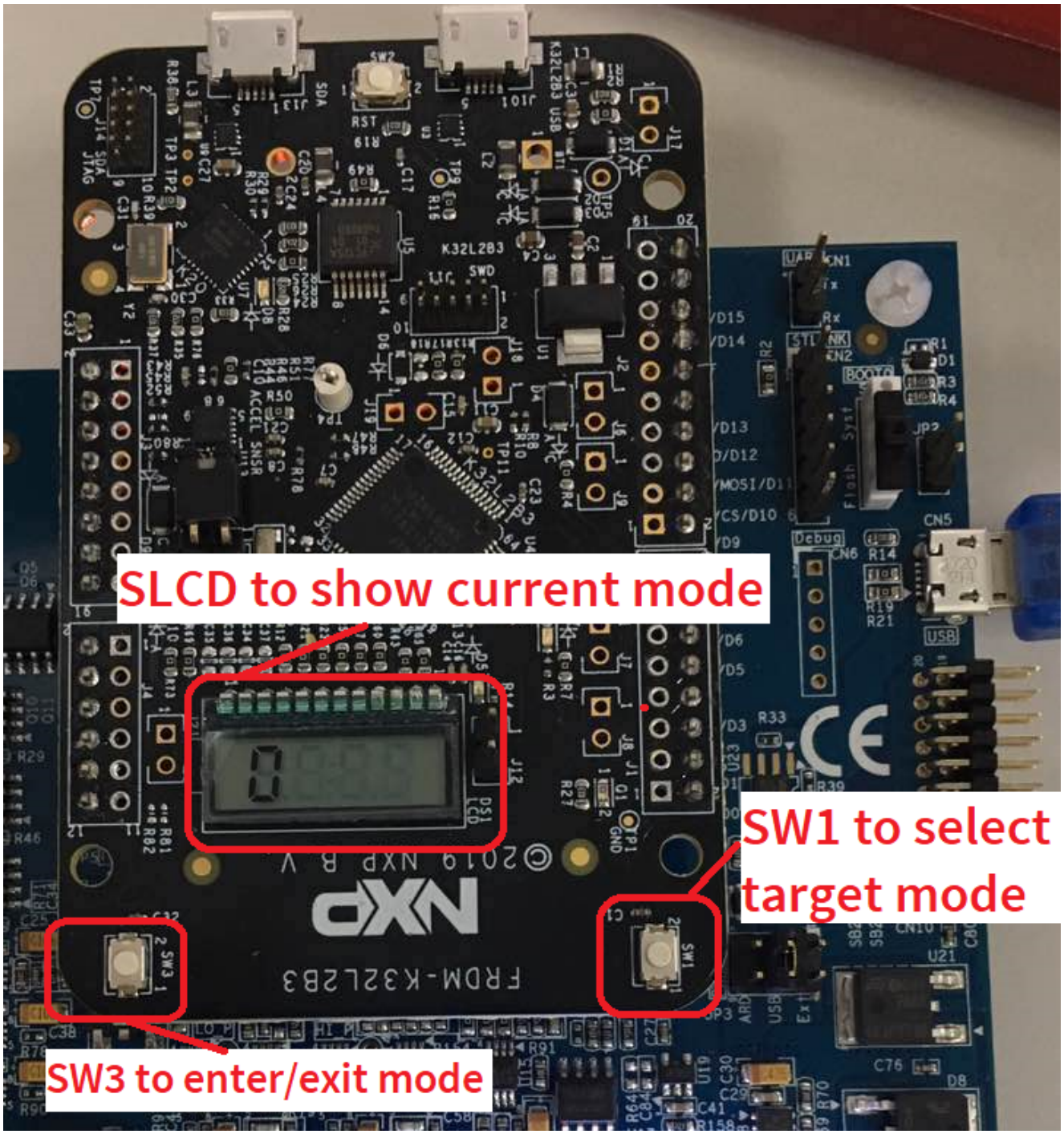


Figure 5. Use button to switch power mode

3.2 Software design

MCUXpresso SDK software package provides the driver for SMC, PMC, LLWU and clock modules. In the application software, we can use these driver APIs to operate the power modes with other peripheral drivers.

3.2.1 Switch power modes with software

Totally 10 power modes are covered in this application demo:

```
/* Power mode definition used in application. */
typedef enum _app_power_mode
{
    kAPP_PowerModeRun, /* Normal RUN mode */
    kAPP_PowerModeWait, /* WAIT mode. */
    kAPP_PowerModeStop, /* STOP mode. */
    kAPP_PowerModeVlpr, /* VLPR mode. */
    kAPP_PowerModeVlpw, /* VLPW mode. */
    kAPP_PowerModeVlps, /* VLPS mode. */
    kAPP_PowerModeLls, /* LLS mode. */
    kAPP_PowerModeVlls0, /* VLLS0 mode. */
    kAPP_PowerModeVlls1, /* VLLS1 mode. */
    kAPP_PowerModeVlls3, /* VLLS3 mode. */
    kAPP_PowerModeMax
} app_power_mode_t;
```

The `APP_PowerModeSwitch()` function is the most important function to execute the power mode switch. It uses the SMC driver's API to control the target power mode:

```
void APP_PowerModeSwitch(smc_power_state_t curPowerState, app_power_mode_t
targetPowerMode)
{
    smc_power_mode_vlls_config_t vlls_config;
    vlls_config.enablePorDetectInVlls0 = true;

    switch (targetPowerMode)
    {
        case kAPP_PowerModeVlpr:
            APP_SetClockVlpr(); /* setup the lower clock source for VLPR. */
            SMC_SetPowerModeVlpr(SMC);
            while (kSMC_PowerStateVlpr != SMC_GetPowerModeState(SMC))
            {
            }
            break;

        case kAPP_PowerModeRun:

            /* Power mode change. */
            SMC_SetPowerModeRun(SMC);
            while (kSMC_PowerStateRun != SMC_GetPowerModeState(SMC))
            {
            }

            /* If enter RUN from VLPR, change clock after the power mode change.
*/
            if (kSMC_PowerStateVlpr == curPowerState)
            {
               APP_SetClockRunFromVlpr(); /* setup the higher clock source for
RUN. */
             }
             break;

        case kAPP_PowerModeWait:
            SMC_PreEnterWaitModes();
            SMC_SetPowerModeWait(SMC);
            SMC_PostExitWaitModes();
```

```
            break;

        case kAPP_PowerModeStop:
            SMC_PreEnterStopModes();
            SMC_SetPowerModeStop(SMC, kSMC_PartialStop);
            SMC_PostExitStopModes();
            break;

        case kAPP_PowerModeVlpw:
            SMC_PreEnterWaitModes();
            SMC_SetPowerModeVlpw(SMC);
            SMC_PostExitWaitModes();
            break;

        case kAPP_PowerModeVlps:
            SMC_PreEnterStopModes();
            SMC_SetPowerModeVlps(SMC);
            SMC_PostExitStopModes();
            break;

        case kAPP_PowerModeLls:
            SMC_PreEnterStopModes();
            SMC_SetPowerModeLls(SMC);
            SMC_PostExitStopModes();
            break;

        case kAPP_PowerModeVlls0:
            vlls_config.subMode = kSMC_StopSub0;
            SMC_PreEnterStopModes();
            SMC_SetPowerModeVlls(SMC, &vlls_config);
            SMC_PostExitStopModes();
            break;

        case kAPP_PowerModeVlls1:
            vlls_config.subMode = kSMC_StopSub1;
            SMC_PreEnterStopModes();
            SMC_SetPowerModeVlls(SMC, &vlls_config);
            SMC_PostExitStopModes();
            break;

        case kAPP_PowerModeVlls3:
            vlls_config.subMode = kSMC_StopSub3;
            SMC_PreEnterStopModes();
            SMC_SetPowerModeVlls(SMC, &vlls_config);
            SMC_PostExitStopModes();
            break;

        default:
            break;
      }
}
```

Entering a new power mode is kind of like switching to a new task with new working condition in OS. Some operations should be done before the entering to close the current mode and prepare for the new mode. When come to a new mode, some initial work should be done as well. So, in the application demo code, the function of `APP_PowerPreSwitchHook()` and `APP_PowerPostSwitchHook()` are created to pack these operations. To make the MCU costing as lower power consumption as possible, the unnecessary peripherals are disabled before entering the wait/stop modes, and recovery after waken up. In the application demo, the UART peripheral for terminal interaction, and the output pins are disabled in low power modes. during the MCU is sleep, only the SLCD driven by OSC32 clock and the NVIC/AWIC are still alive.

3.2.2 Use on-board buttons to select mode

SW1 (PTA4) and SW3 (PTC3) are used to enter the selected target mode and exit. They are driven totally by ISR:

```
/* PTC3. */
void APP_WAKEUP_BUTTON_IRQ_HANDLER(void)
{
    if ((1U  <<APP_WAKEUP_BUTTON_GPIO_PIN) &
PORT_GetPinsInterruptFlags(APP_WAKEUP_BUTTON_PORT))
    {
        /* Disable interrupt. */
        PORT_SetPinInterruptConfig(APP_WAKEUP_BUTTON_PORT,
APP_WAKEUP_BUTTON_GPIO_PIN, kPORT_InterruptOrDMADisabled);
        PORT_ClearPinsInterruptFlags(APP_WAKEUP_BUTTON_PORT, (1U <<
APP_WAKEUP_BUTTON_GPIO_PIN));
        sw3_irq_done = true;
    }
}

/* PTA4. */
void PORTA_IRQHandler(void)
{
    uint32_t flags = PORT_GetPinsInterruptFlags(PORTA);
    PORT_ClearPinsInterruptFlags(PORTA, flags); /* clear flags. */

    /* PTA4. */
    if ( 0u != ((1u << 4u) & flags) )
    {
        sw1_irq_counter = (sw1_irq_counter+1u)% 10u;

        slcd_set_number(0u, sw1_irq_counter, false);
        PRINTF(".");
    }
}

app_power_mode_t APP_GetTargetPowerMode(void)
{
    sw3_irq_done = false;

    /* enable the pin detection. */
    PORT_SetPinInterruptConfig(PORTA, 4u, kPORT_InterruptFallingEdge);

    /* wait for a new sw3_irq. */
    PORT_SetPinInterruptConfig(PORTC, 3u, kPORT_InterruptFallingEdge);
    while (!sw3_irq_done)
    {}
    /* disable the pin detection. */
    PORT_SetPinInterruptConfig(PORTA, 4u, kPORT_InterruptOrDMADisabled);

    PRINTF("%d\r\n", sw1_irq_counter);

    return (app_power_mode_t)(sw1_irq_counter);
}
```

SW1 is only activated during the APP_GetTargetPowerMode() function is running. After the SW3 is pressed in this function, the SW1 is locked again. Then the value increased by SW1 previously would be return as the selection.

SW3 is also used as the wakeup source:

- When the NVIC is still alive, SW3 use port interrupt to wakeup the MCU and exit the low power modes.

- When the NVIC is down in some ultra low power modes, SW3 is routed to the LLWU, which can wakeup the MCU through AWIC.

3.2.3 Keep memory alive in low-power modes

To show if the memory can still keep the data in various power modes. A variable with software token written inside is used to indicate the content is lost or not. In the application demo, a token, `APP_LOW_POWER_MEM_TOKEN`, is written to variable of `app_always_keep_value` before entering a low power sleeping mode, then read it after the MCU is waken up again. For the low-power modes that need the reset routine to wakeup, the software also will read the token variable before writing a new token. Everytime when the MCU is waken up, either from reset or inplace, the software would read the token variable and compare it with the expected value, then tell the user on the SLCD screen and UART terminal.

```
#define APP_LOW_POWER_MEM_TOKEN 0x55555555

volatile uint32_t app_always_keep_value; /* keep the token value during in low
power modes. */

int main(void)
{
    ...

    /* Unlock all the power modes of chip. */
    SMC_SetPowerModeProtection(SMC, kSMC_AllowPowerModeAll);

    /* Clear the low power lock bit. */
    if (kRCM_SourceWakeup & RCM_GetPreviousResetSources(RCM)) /* Wakeup from
VLLS. */
    {
        PMC_ClearPeriphIOIsolationFlag(PMC);
        NVIC_ClearPendingIRQ(LLWU_IRQn);
        PRINTF("\r\nMCU wakeup from VLLS modes...\r\n");

        if (APP_LOW_POWER_MEM_TOKEN == app_always_keep_value)
        {
            slcd_set_number(2, 1, false);
            PRINTF("memory value is kept.\r\n");
         }
        else
        {
            slcd_set_number(2, 0, false);
            PRINTF("memory value is missed.\r\n");
         }
    }
    else
    {
        PRINTF("\r\npower mode switch example.\r\n");
    }

    while (1)
    {
        ...

        /* Wait for user response */
        targetPowerMode = APP_GetTargetPowerMode();

        /* only go next with avaiable and right input. */
        if (1u == APP_CheckPowerMode(curPowerState, targetPowerMode))
        {
            APP_PowerPreSwitchHook(curPowerState, targetPowerMode);
```

```
            /* setup wakeup source. */
            if ( (kAPP_PowerModeRun != targetPowerMode)
                && (kAPP_PowerModeVlpr != targetPowerMode) )
            {
                APP_SetWakeupConfig(targetPowerMode);
             }

             APP_PowerModeSwitch(curPowerState, targetPowerMode);
             APP_PowerPostSwitchHook(curPowerState, targetPowerMode);
        }
    }

static void APP_PowerPreSwitchHook(smc_power_state_t originPowerState,
app_power_mode_t targetMode)
{
    /* setup a token in memory. */
    app_always_keep_value = APP_LOW_POWER_MEM_TOKEN;
    slcd_set_number(2, SLCD_ON_SHOW_NUMBER_NONE, false);
    PRINTF("Set a token in memory\r\n");

    ...
}

static void APP_PowerPostSwitchHook(smc_power_state_t originPowerState,
app_power_mode_t targetMode)
{
    /* check the token. */
    if (APP_LOW_POWER_MEM_TOKEN == app_always_keep_value)
    {
        slcd_set_number(2, 1, false);
        PRINTF("token value is kept.\r\n");
    }
    else
    {
        slcd_set_number(2, 0, false);
        PRINTF("token value is missed.\r\n");
    }

    ...
}
```

3.3 Run application demo project

Finally, after building the project and downloading the image to FRDM-K32L2B board, the application demo can run to measure the working current in different power mode. The UART terminal can output the log information.

Let us try the multimeter first. Just as mentioned previously, put the multimeter (in current measurement mode) into the series connection of J20. Connect the on-board debugger to PC and open the terminal tool for UART communication (9600, 8, N), as shown in Figure 6.

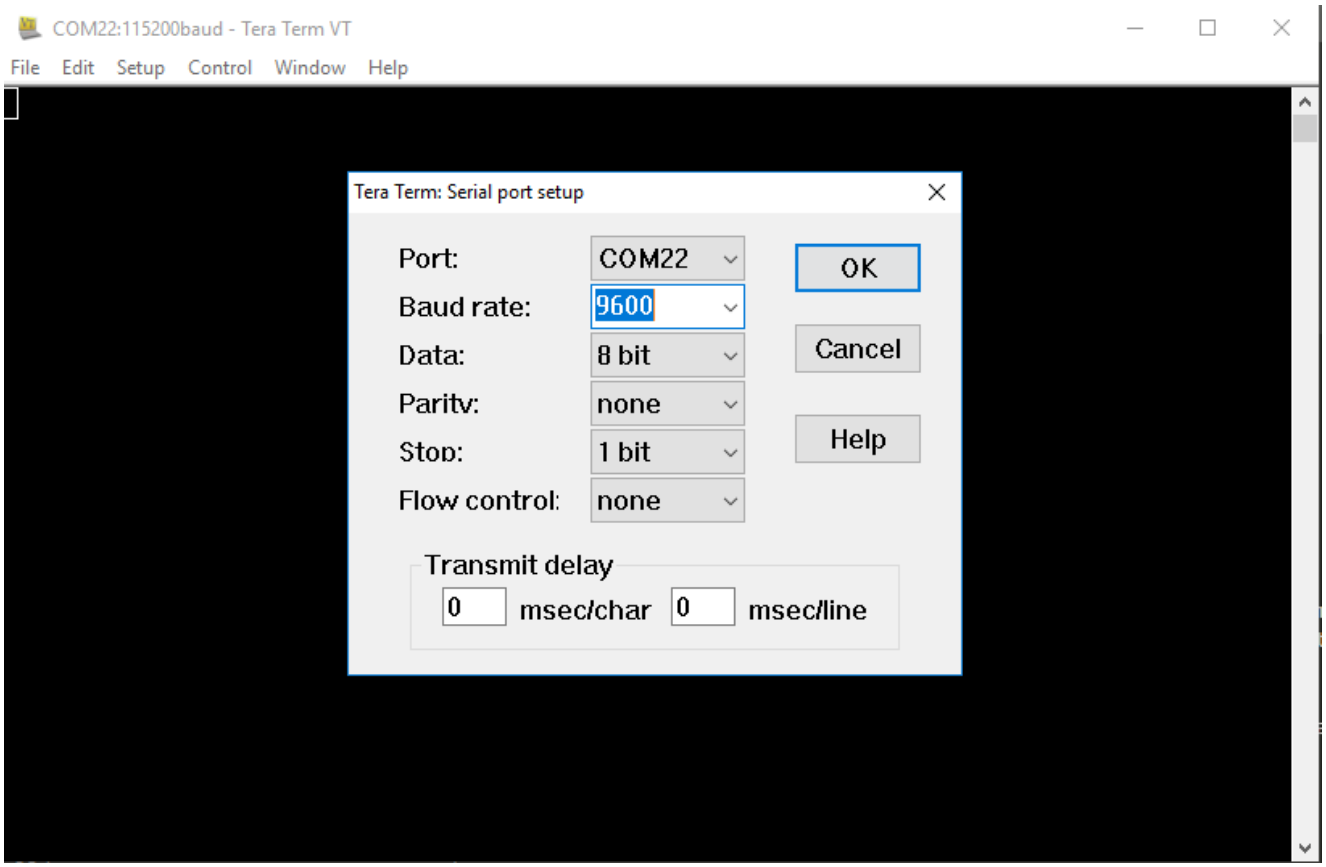


Figure 6. UART terminal settings

Now the application demo is running.

The initial power mode is **RUN**. For the **RUN** mode, the SLCD shows **0** , as shown in Figure 7.

Figure 7. RUN mode

The UART terminal also shows the menu of the power mode selection, as shown in Figure 8.

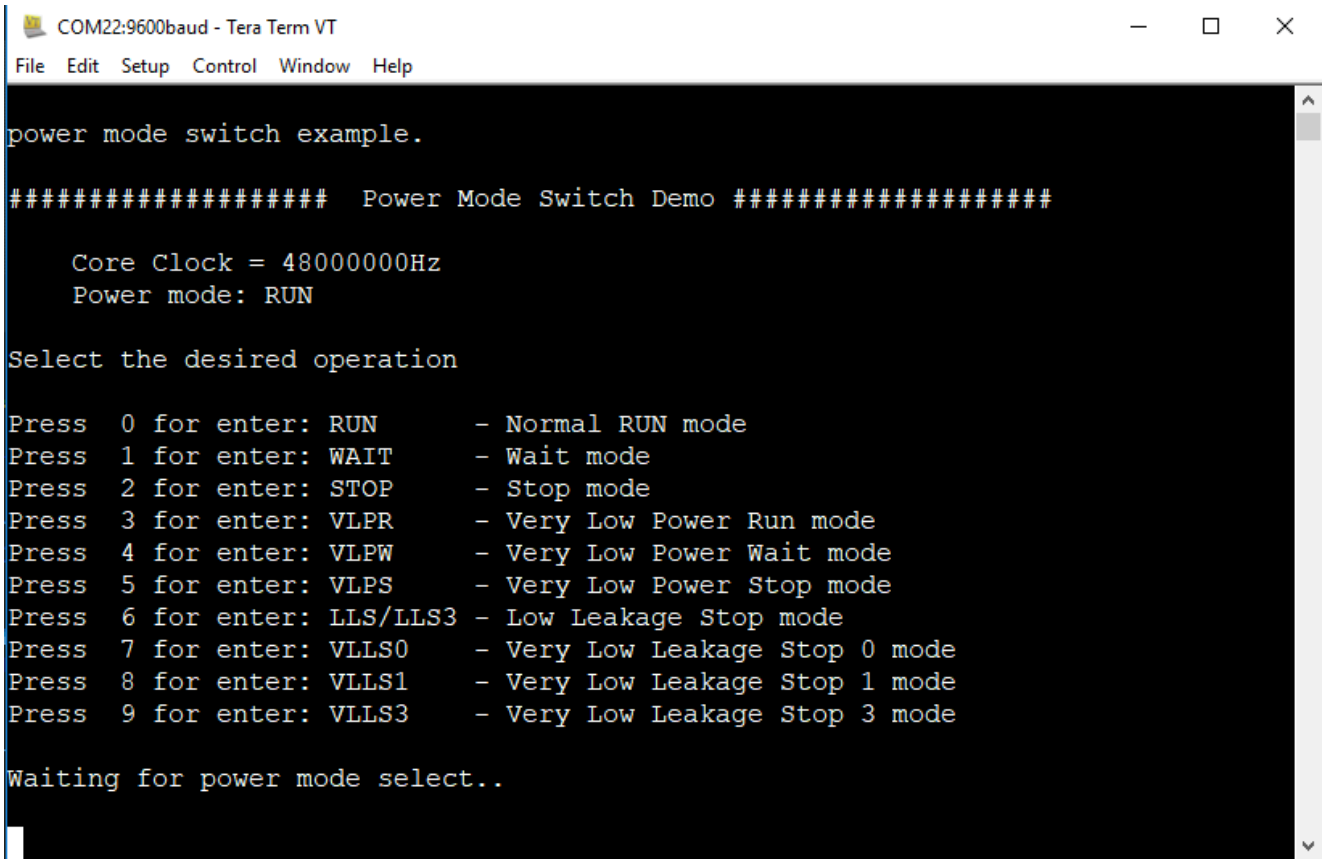


Figure 8. Power mode selection on UART terminal

Then press **SW1** to increase the number, which indicates the power mode in the menu. For each press, the number showing on the SLCD increases one by one. Also, a **dot** shows in the UART terminal. For example, **2** is for the **STOP** mode.

Once the selection is done, press **SW3** to enter the target mode. The SLCD shows a colon to tell that the MCU is sleeping, as shown in Figure 9. In such a case, pressing **SW1** gets no responses.

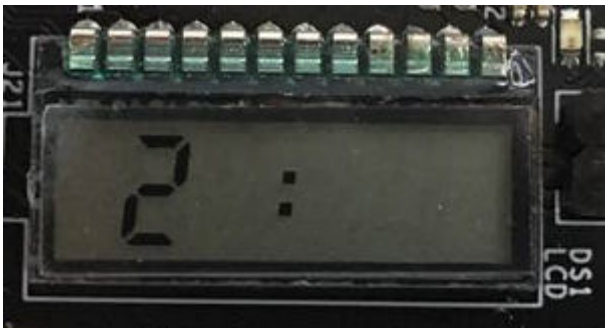


Figure 9. STOP mode

Press **SW1** again to wake up the MCU. Then the number on the SLCD change to **0** and **1**, as shown in Figure 10. The MCU return to **RUN** with software token available.

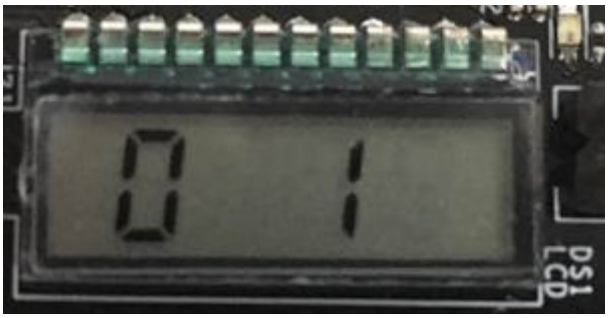


Figure 10. RUN mode with software token available

NOTE

The colon disappears and it means that the MCU is now running with no sleep.

0 on th left means the MCU just returns to the RUN mode. It is **3** when the MCU is waken up from VLPS/VLPW to VLPR.

1 on the right means the token in the variable is still kept. It is **0** when the token value is missed.

There is only one exception for the VLLS0 (code 7). The SLCD shows nothing when the MCU is sleeping. This mode powers down the SLCD mode. But after **SW3** is pressed, the MCU is waken up and the SLCD is back on line again.

Press **SW1** to select other power mode and press **SW3** to enter or exit the target modes.

During the operations to switch power modes, users can read the current value on the multimeter, which is showing the real-time power consumption.

4 Conclusion

Per running this application demo, we measure the power consumption condition of K32L2B on FRDM-K32L2B board. Table 4 displays all the measuring values.

NOTE

Even in the ultra-low-power modes, the LLWU, SLCD with OSC32 clock source are still active, as they are specially designed for low-power usage.

Table 4. Power consumption in various power modes of K32L2B

Power Mode	VDD_I	Memory kept	Comment
RUN	6.04 mA	Yes	48 MHz CORE clock. UART enabled.
WAIT	3.23 mA	Yes	CORE sleep. UART disabled. NVIC wakeup.
STOP	0.16 mA	Yes	CORE sleep. UART disabled. NVIC wakeup.
VLPR	0.21 mA	Yes	4 MHz CORE clock. UART enabled.
VLPW	0.10 mA	Yes	CORE sleep. UART disabled. NVIC wakeup.
VLPS	8.7 uA	Yes	CORE sleep. UART disabled. NVIC wakeup.
LLS	8.4 uA	No	CORE sleep. UART disabled. LLWU wakeup. SLCD & OSC32 disabled.
VLPS0	1.2 uA	No	CORE sleep. UART disabled. LLWU wakeup. SLCD & OSC32 enabled.
VLLS1	7.2 uA	No	CORE sleep. UART disabled. LLWU wakeup. SLCD & OSC32 enabled.
VLLS3	7.7 uA	No	CORE sleep. UART disabled. LLWU wakeup. SLCD & OSC32 enabled.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 02/2020

Document identifier: AN12736

arm