Introduction

In this hands-on lab, we will learn how to use the Low Power modes on a K22F using the Low Power Manager in the Freescale MQX BSP.

Resources

PC with the following software:
• IAR IDE Version 7.20 or later
• MQX Software, Release 4.1 for FRDMK22F120M
Hardware:
• FRDM-K22F
• USB Type A-micro B Cable

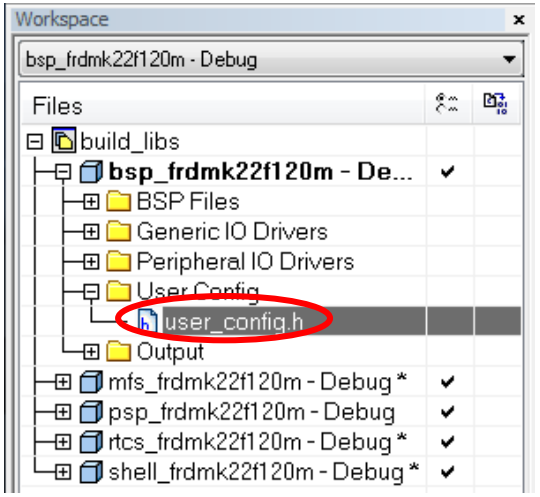
Lab 1: Low Power Demo

1.1 In this lab, we will run one of the example projects that ships with **MQX Version 4.1.0 for FRDMK22F120M.**

1.2 Make the following connection to the Freedom board:
- USB type A-miniB cable between the PC and the FRDM-K22F.

1.3 In Embedded Workbench v7.20 select [File | Open | Workspace] and choose C:\Freescale\Freescale_MQX_4_1_FRDMK22F120M\build\frdmk22f120m\iar\build_libs.eww

1.4 In the project "bsp_frdmk22f120m" open the folder "User Config" and open the file user_config.h



1.5 Enable the low power manager. Set this define to a 1.
```
#define MQX_ENABLE_LOW_POWER    1
```

1.6 Then build the bsp project. Right click on "bsp_frdmk22f120m" and choose Make or press[F7].

1.7 Next build the psp project. Right click on "psp_frdmk22f120m" and choose Make. (Do not build the other library projects. It is unnecessary at this time and takes several minutes.)

1.8 Now create a new empty workspace. Select [File | New | Workspace].

1.9 Add the existing example project to this workspace. Select [Project | Add Existing

Project…] and browse to the project lowpower_frdmk22f120m.ewp in folder: C:\Freescale\Freescale_MQX_4_1_FRDMK22F120M\mqx\examples\lowpower\iar\lowpower_frdmk22f120m

1.10 Save the new workspace with [File | Save Workspace]. Save it to any location with a name that you like.

1.11 Make the project by clicking the Make button.

1.12 Flash the image to the board using the built-in flash-programmer; do this by running a debugging session with keyboard shortcut [Ctrl-D]. Run the code with [F5].

1.13 Next, immediately terminate the debug session (left click the red 'X') and power cycle your board by unplugging the USB connection to the board and replacing the USB connection.

1.14 Open Tera Term. Click Setup->Serial Port and choose your mbed Serial Port [COMxx] for the port (the serial port number will vary from system to system; check your USB devices for this number). Then, select 115200 for the baud rate, 8 bit for the data configuration, no parity, 1 stop bit, and no flow control and click OK.

1.15 Reset the Freedom board and observe the output from the printf statements.

1.16 Example Stage 1: In the Tera Term window, you will see that the application reports that it is in HSRun mode. Follow the on-screen instructions to move to the next stage. [**NOTE:** The button the application refers to is SW3 on your Freedom board.)

```
MQX Low Power Modes Demo

********************************************************************************
************** Operation mode : LPM_OPERATION_MODE_HSRUN ***********************
********************************************************************************
Core Low Power Settings:
  Mode:       LPM_CPU_POWER_MODE_HSRUN
Wake up events:
  LLWU_PE1 = 0x00 Mode wake up events from pins 0..3
  LLWU_PE2 = 0x00 Mode wake up events from pins 4..7
  LLWU_PE3 = 0x00 Mode wake up events from pins 8..11
  LLWU_PE4 = 0x00 Mode wake up events from pins 12..15
  LLWU_ME  = 0x00 Mode wake up events from internal sources

Info: HSRUN operation mode is mapped on HSRUN power mode by default.
      The core runs at full clock speed.
      It continues the execution after entering the mode.
      LED2 blinks quickly. LED1 toggles after the button press.
Press button to move to next operation mode.
```

1.17 Example Stage 2: The application first moves to normal RUN mode and then disables the *idle task sleep* feature. The idle task is allowed to run for 1 second and the number of idle task loops are counted. Then it does the same thing with the idle task sleep feature enabled, showing the number of loops is dramatically decreased because the system is sleeping for most of the time instead of wasting time looping.

```
Idle task sleep feature disabled.
Task suspended for 1 second to let run the idle task.
Idle loops per second with idle sleep disabled: 1590325
Idle task sleep feature enabled.
Task suspended for 1 second to let run the idle task.
Idle loops per second with idle sleep enabled:  63
Idle task sleep feature disabled.

Press button to move to next operation mode.
```

1.18 Example Stage 3: Press SW3, it changes the clock configuration to 2 MHz and then transitions into Wait state which is mapped to Very Low Power Run (VLPR) on the MCU.

```
*****************************************************************************
************** Operation mode : LPM_OPERATION_MODE_WAIT **********************
*****************************************************************************
Core Low Power Settings:
  Mode:      LPM_CPU_POWER_MODE_VLPR
Wake up events:
  LLWU_PE1 = 0x00 Mode wake up events from pins 0..3
  LLWU_PE2 = 0x00 Mode wake up events from pins 4..7
  LLWU_PE3 = 0x00 Mode wake up events from pins 8..11
  LLWU_PE4 = 0x00 Mode wake up events from pins 12..15
  LLWU_ME  = 0x00 Mode wake up events from internal sources

Info: WAIT operation mode is mapped on VLPR power mode by default.
      It requires 2 MHz core clock and bypassed pll.
      Core continues the execution after entering the mode.
      LED2 blinks slowly, LED1 toggles after the button press.

Changing frequency to 2 MHz.

Setting operation mode to LPM_OPERATION_MODE_WAIT ... OK

Press button to move to next operation mode.
```

1.19 Example Stage 4: Press SW3 again, now it transitions into Sleep state which is mapped to Wait mode on the MCU. In this part of the example, the Low Power Timer is enabled to trigger an interrupt in 10 seconds which wakes the system up.

```
*****************************************************************************
************** Operation mode : LPM_OPERATION_MODE_SLEEP **********************
*****************************************************************************
Core Low Power Settings:
  Mode:      LPM_CPU_POWER_MODE_WAIT
Wake up events:
  LLWU_PE1 = 0x00 Mode wake up events from pins 0..3
  LLWU_PE2 = 0x00 Mode wake up events from pins 4..7
  LLWU_PE3 = 0x00 Mode wake up events from pins 8..11
  LLWU_PE4 = 0x00 Mode wake up events from pins 12..15
  LLWU_ME  = 0x00 Mode wake up events from internal sources

Info: SLEEP operation mode is mapped on WAIT power mode by default.
      The core is inactive in this mode, reacting only to interrupts.
      The LPM_CPU_POWER_MODE_FLAG_SLEEP_ON_EXIT is set on Kinetis, therefore
      core goes to sleep again after any isr finishes. The core will stay awake
      after call to _lpm_wakeup_core() from timer wakeup or serial interrupt.
      LED2 doesn't blink, LED1 toggles after the button press.

Timer wakeup set to +10 seconds.

Setting operation mode to LPM_OPERATION_MODE_SLEEP ... OK

Core woke up by timer wakeup.

Press button to move to next operation mode.
```

1.20 Press SW3 again and the application transitions to the STOP mode which is mapped to the MCU's LLS mode. The Low Power Timer is again enabled to trigger an interrupt in 10 seconds which wakes the system up.

```
*****************************************************************************
************** Operation mode : LPM_OPERATION_MODE_STOP **********************
*****************************************************************************
Core Low Power Settings:
  Mode:      LPM_CPU_POWER_MODE_LLS
Wake up events:
  LLWU_PE1 = 0x00 Mode wake up events from pins 0..3
  LLWU_PE2 = 0x00 Mode wake up events from pins 4..7
  LLWU_PE3 = 0x00 Mode wake up events from pins 8..11
  LLWU_PE4 = 0x00 Mode wake up events from pins 12..15
  LLWU_ME  = 0x01 Mode wake up events from internal sources

Info: STOP operation mode is mapped to LLS power mode by default.
      Core and most peripherals are inactive in this mode, reacting only to
      specified wake up events. The events can be changed in BSP (init_lpm.c).
      Serial line is turned off in this mode. The core will wake up from
      timer wakeup interrupt.
      LED2 doesn't blink, LED1 toggles after the button press.

Timer wakeup set to +10 seconds.

Setting operation mode to LPM_OPERATION_MODE_STOP ...

Core is awake. Moved to next operation mode.
Press button to move to next operation mode.
```

1.21 Press SW3 again, the example starts over.

Code Analysis

1.22 Open the Source folder and `main.c` file.

1.23 The application example creates two tasks:

- **main**: task sets up the BUTTON and LED1 pins and installs the timer and wakeup interrupt. This task manages the operation mode and user interface. In some modes, the timer wakeup interrupt or SCI RX interrupt needs to occur to wake up the core.
- **for_loop_led_task**: sets up the LED2 and toggles its state with a frequency given by current clock configuration. You will see a change of the clock configuration as a change of LED2 blink rate. When the LED2 is not blinking the core is in the low power stop mode and is not executing any task.

1.24 The LWGPIO driver is used for the LED and BUTTON pins handling. The `button_led_init()` function initializes the corresponding pins for input and output and also it installs the falling-edge interrupt for the button (with internal pull-up resistor enabled). The ISR handler toggles the LED2 and sets the event to trigger an action in the main task.

1.25 Find the infinite while loop inside the main task. The application first gets the current operation mode (HSRun in this case) and then waits for a button press. Upon a button press, the device transitions to normal RUN mode using the `_lpm_set_operation_mode` API call.

1.26 After transitioning to RUN, the core sleep feature is enabled/disabled using function `_lpm_idle_sleep_setup()`.The example shows that the number of loops in idle task gets minimized when this feature is enabled, because the idle task puts the core into sleep between interrupts (system tick).

1.27 In the next stage of the example, the core clock frequency is reduced to 2MHz to reduce power consumption, but the core is still executing. The LPM updates the serial driver to change the UART baud rate dividers, allowing the baud rate to stay the same.

```
_lpm_set_clock_configuration(BSP_CLOCK_CONFIGURATION_2MHZ)
```

1.28 The operation modes are set using the Low-power Manager API function `_lpm_set_operation_mode()`. The parameter of this function is one of the LPM constants that correspond to predefined operation modes.

```
_lpm_set_operation_mode (LPM_OPERATION_MODE_WAIT)
```

1.29 The example then waits for a button press, and then it changes the clock back to the default BSP frequency. This time the example changes to the LPM Sleep operation mode, which is mapped to the Kinetis Wait mode. In this mode, the core stops executing and waits for an event to execute again. The application uses the UART and low power timer (LPTMR) interrupts as events to wake-up the core from some low

power operation modes. For the serial line, the interrupt is available only if an interrupt-driven driver is used "ittyb:". The LPTMR wakeup is always set to time "now + 10 seconds". The core is woken up via the LPTMR interrupt and immediately executes the LPTMR ISR. The _lpm_wakeup_core() function is used here to let the core to run also after the ISR finishes. Refer to Kinetis documentation for more information about specific CPU behavior in various low power modes.

```
static void timer_wakeup_isr
    (
        void   *parameter
    )
{
    /* Stop the timer */
    hwtimer_stop(&lpttimer);

    /* Do not return to sleep after isr again */
    _lpm_wakeup_core ();

    /* Signal the timer event */
    _lwevent_set (&app_event, TIMER_EVENT_MASK);
}
```

1.30 Last mode; LPM Stop mode. It is mapped to the LLS mode. The UARTs are disabled in this mode. The LLWU is set to allow the LPTMR to wake up the MCU. The example uses the LPTMR driver to set a 10 second IRQ, and then exits Stop mode. After waking, the UARTs are enabled again, and after a button press, the example starts this sequence over again.

```
static void set_timer_wakeup
    (
        void
    )
{
    hwtimer_start(&lpttimer);
    printf ("\nTimer wakeup set to +10 seconds.\n");
}
```

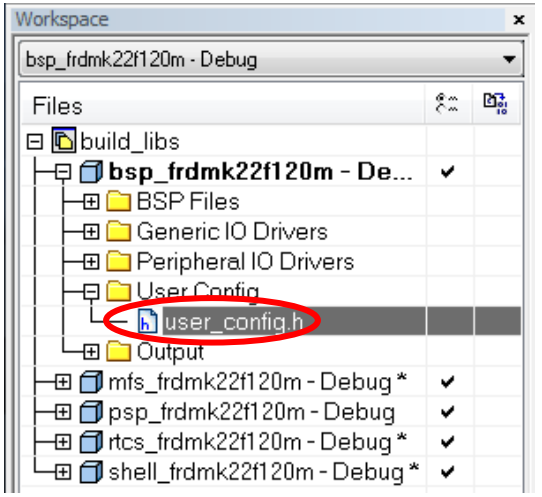
Lab 2: The challenge in the Low Power Demo

1.31 This lab explains how to modify the BSP to enable the LPM to allow a press of **SW2** on the FRDM-K22F to wake up the MCU from MQX Sleep mode and the UART interrupt to wake up the MCU from Stop mode.

1.32 The BSP has enabled the serial driver in polling mode. The LPM_OPERATION_MODE_SLEEP requires an interrupt to wake up. To allow the UART to wake up the MCU it is just necessary to enable the "ittyb:" driver in the BSP.

1.33 In Embedded Workbench v7.20 open the workspace located in C:\Freescale\Freescale_MQX_4_1_FRDMK22F120M\build\frdmk22f120m\iar\build_libs.eww

1.34 In the project "bsp_frdmk22f120m" open the folder "User Config" and open the file user_config.h



1.35 Change these two lines in the user_config.h file as follow:

```
#define BSPCFG_ENABLE_TTYB          0
#define BSPCFG_ENABLE_ITTYB         1
```

1.36 Add this line in the user_config.h file:
#define BSP_DEFAULT_IO_CHANNEL "ittyb:"
This is all needed to enable the UART driver with interrupts.

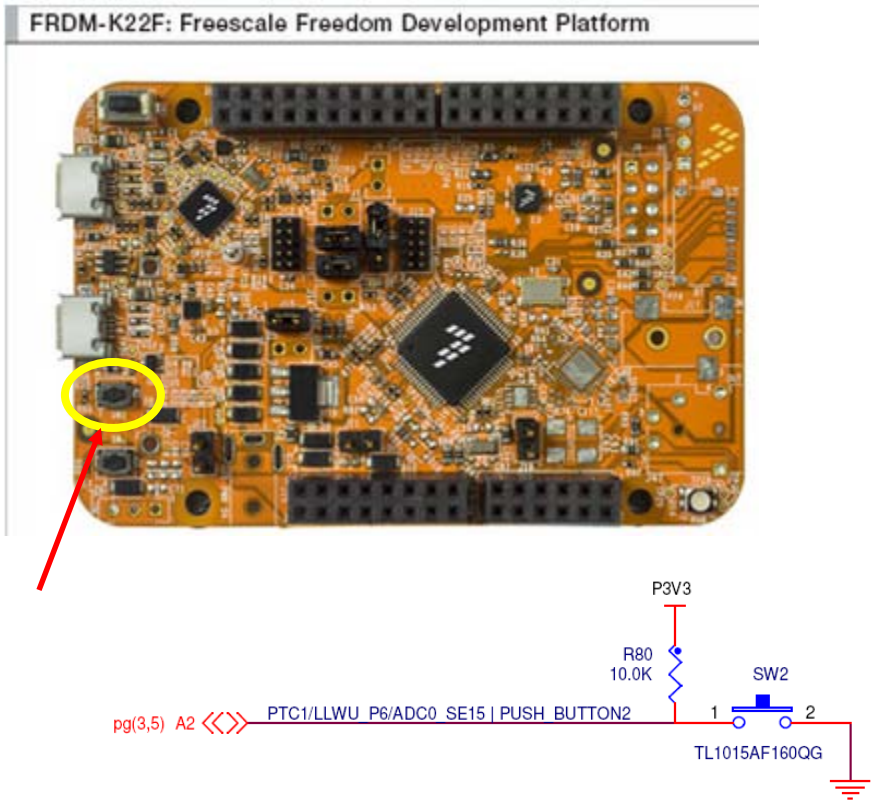
1.37 To enable the SW2 it is necessary to include this source of wake up for the Stop mode in the LPM_CPU_OPERATION_MODES structure.

1.38 There are two kinds of wake up sources:
- From an external pin.
- From an internal module (like LPTMR interrupt).

1.39 The SW2 in the FRDM-K22F is connected to the port pin PTC1. It is necessary to then enable the

wake up source for the LLWU inputs the PTC1 which is the LLWU_P6 according with the “Wake up sources for LLWU inputs” table 3-14 from the Reference Manual for the K20 family: K22P121M120SF7RM.pdf

LLWU_P6	PTC1/LLWU_P6 pin



1.40 Open `init_lpm.c` which is located in BSP_files folder.

1.41 Based on previous step. In the `LPM_CPU_OPERATION_MODES` structure we can find that for the `LPM_OPERATION_MODE_STOP` array there is only the mask to enable the LPTMR as a wakeup source for the MQX STOP mode. Add the mask for the pin LLWU_P6 to the array (this should be replace the 0 in the 4th element as shown below).

```
// LPM_OPERATION_MODE_STOP
{
    LPM_CPU_POWER_MODE_LLS,  // Index of predefined mode
    0,                       // Additional mode flags
    0,                       // Mode wake up events from pins 0..3
    LLWU_PE2_WUPE6_MASK,     // Mode wake up events from pins 4..7
    0,                       // Mode wake up events from pins 8..11
    0,                       // Mode wake up events from pins 12..15
    LLWU_ME_WUME0_MASK       // Mode wake up events from internal in
},
```

1.42 Make the project with keyboard shortcut [F7].

1.43 Now close the build_libs workspace and reopen the lowpower_example project.

1.44 Comment out the preprocessor error statement at line #35 of main.c (it is no longer valid given the modifications we have just made)

```
#ifndef BSP_DEFAULT_IO_CHANNEL_DEFINED
//#error This application requires BSP_DEFAULT_IO_CHA
#endif
```

1.45 Rebuild the lowpower_example project using the keyboard shortcut [F7].

1.46 Flash and run the code as we did in steps 1.12 – 1.15 (remember to disconnect or close Tera Term before power cycling your Freedom board). What happens?