# Table of Contents

- Objectives ................................................................................................................. 3
- Hardware ...................................................................................................................... 3
- Software ...................................................................................................................... 3
- Lab description ............................................................................................................ 3
- Power operating ......................................................................................................... 4
  - Power rails ............................................................................................................... 4
  - LDO_ENABLE ........................................................................................................... 4
  - External power supplies using Power Management IC (PMIC) ................................. 4
- Power configuration .................................................................................................. 5
  - Active mode ............................................................................................................ 5
  - Sleep mode .............................................................................................................. 6
  - Deep Sleep mode .................................................................................................... 6
  - Deep Power Down mode ........................................................................................ 6
  - Full Deep Power Down mode .................................................................................. 6
- Wake up source ......................................................................................................... 6
  - Timers function during deep sleep and deep power down ....................................... 6
  - Peripherals to wake from deep sleep ...................................................................... 7
  - Wake up process ..................................................................................................... 8
- RAM partitions and power consumption ................................................................... 8
- Core operation in Active Mode .................................................................................. 9
  - Maximum core operation frequencies and minimum core voltage in Active Mode 9
- Body Bias ................................................................................................................ 9
- Power Lab 1 – Cortex M33 Operating frequency at 0.7V VDDCore ......................... 10
  - MCUXpresso SDK power_manager example ......................................................... 10
  - PCA9420 and powerlib API ................................................................................... 12
  - Core Frequency .................................................................................................... 14
  - MCUXpresso SDK power_manager example ......................................................... 15
- Serial Terminal ......................................................................................................... 16
- Build the project ...................................................................................................... 17
- Run the project ....................................................................................................... 18
Power Lab 2 – Wake from deep sleep using RTC, Pin Interrupt and USART .......... 21
  RTC wake (high resolution counter) as wake up source ........................................... 21
  Pin Interrupt as wake up source ..................................................................................... 22
  USART as another wake up source .................................................................................. 23
  Understanding deep sleep parameters that passes to deep sleep power API ........ 24
  Build the project ............................................................................................................. 25
  Run the project ............................................................................................................... 25
  Deep sleep wake by RTC .................................................................................................. 27
  Deep sleep wake by Pin .................................................................................................... 28
  Deep sleep wake by USART ............................................................................................ 29
Power Lab 3 – Using RTC to wake from deep power down ........................................ 30
  Build the project ............................................................................................................. 30
  Run the project ............................................................................................................... 30
  Deep power down wake by deep sleep .......................................................................... 31
  Deep power down wake by reset .................................................................................... 32
  Measuring power consumption on VDDCore JP29 ....................................................... 35
Objectives
In this lab, you will learn:
- Utilize the Powerlib for PMIC
- Low power modes
- Operating modes

Hardware
- NXP MIMXRT685-EVK Rev E
- Micro-USB cable
- Digital Multimeter
- Jumper wire (Female-to-female)
- One pin header (Male-to-male)

Software
- Download the latest MCUXpresso IDE version 11.4.0 and above at [https://mcuxpresso.nxp.com](https://mcuxpresso.nxp.com)
- Download the latest SDK 2.10.0 and above for EVK-MIMXRT685 at [https://mcuxpresso.nxp.com](https://mcuxpresso.nxp.com)
- Power management labs
- Serial Terminal Console such as Putty, minicom, TeraTerm, etc.

Lab description
This lab shows user how to use the RT685 powerlib API functions to achieve desired operating mode and low power modes.
Power operating

Power rails

There are four power rails:

- 1.8V Always On - VDD_A01V8
- 3.3V for IOs USB, and System - VDDIO0, VDDIO1, VDDIO2, USB1_VDD3V3, VDDA_BIAS
- 1.8V for IOs and System - VDDIO0, VDDIO1, VDDIO2, VDD1V8, VDDA_ADC1V8, VDDA_BIAS
- VDDCORE.

Only RTC wakeup timers, LDO_ENABLE, PMC_PMIC (interrupt and mode pins) and Reset Pin are on VDD_A01V8 “Always On” domain when voltage (1.71 – 1.89V) is supplied.

LDO_ENABLE

LDO_ENABLE pin is used to select whether VDDCore is using internal LDO or from external. When LDO_ENABLE pin is pulled high, VDDCore will be supplied from internal LDO and the LDO is powered from the VDD1V8. When LDO_ENABLE pin is pulled low, VDDCore will be supplied from external supply.

External power supplies using Power Management IC (PMIC)

Using external PMIC provides flexibility to configure the power supply rails as application needs. For example, uSDHC may change the VDDIO rail from 3.3V to 1.8V, or deep sleep power rail configuration.

RT685 has a dedicated I²C pins to communicate with PMIC, this allows different power configuration for low power modes or different operating mode at certain core voltage. RT685 allows PMIC to wake CM33 from PMIC_IRQ interrupt pin. The two PMIC_MODE pins allows CM33 to enter pre-configured PMIC power configuration without going through setup every time via I²C.
Power configuration
There are four type of power modes in RT6xx: Active, Sleep, Deep Sleep, Deep Power Down and Full Deep Power Down. The external PMIC referencing the PMIC_MODE pins from RT685 to provide appropriate power configurations.

Table 1 PMIC Mode Pins and Low Power modes

<table>
<thead>
<tr>
<th>PMIC_MODE[1:0]</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low Power Modes</td>
<td>Active / Sleep</td>
<td>Deep Sleep</td>
<td>Deep Power Down</td>
<td>Full Deep Power Down</td>
</tr>
</tbody>
</table>

Active mode
The power consumption in Active mode is determined by PSCCTL, PDRUNCFG, clock sources, peripheral clocks and/or CPU operating modes.
Sleep mode
Clock to the CM33 is gated, execution of instruction is suspended until interrupt occurs. Memories, clocks, and peripherals can be operation, and there are determined by PSCCTL, PDRUNCFG, clock sources, peripheral clocks. Any interrupt can wake up the CM33.

Deep Sleep mode
main_clk and CM33 will be gated to be in Deep Sleep. Memories, certain clocks and certain peripherals can be active are determined by PDSLEEPCFG.

Deep Power Down mode
VDDCore, memories, clocks and peripherals are shutoff except RTC “Always on” Domain. RTC wake-up timers, reset or PMIC interrupt can be used to wake up the system.

Full Deep Power Down mode
VDDCore, VDD1V8, 3V3 power rail, memories, clocks and peripherals are shutoff except RTC on Always on Domain. RTC wake timers, reset or PMIC interrupt can be used to wake up the system.

Wake up source
There are many peripherals that can be served as the wake-up function for the deep sleep low power mode. (Refer to STARTEN0 and STARTEN1 of SYSCTL0 in RT685 User Manual)

Timers function during deep sleep and deep power down
Table 2 shows timers’ comparison in deep sleep, it also shows which clock source it requires to stay active. RTC is the only timer active during deep power down or full deep power down.
### Table 2 Timers comparison in low power modes

<table>
<thead>
<tr>
<th>Timer</th>
<th>Counter Resolution</th>
<th>Deep Sleep</th>
<th>Deep Power Down</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read back counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Elapsed time)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Watchdog Timer (WDT)</td>
<td>24-bit</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Micro-tick Timer (uTick)</td>
<td>31-bit</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multi-Rate Timer (MRT)</td>
<td>24-bit</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Counter Timer (Ctimer)</td>
<td>32-bit</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SC Timer</td>
<td>16- or 32-bit</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real_Time Clock (RTC)</td>
<td>15- (subsec) or</td>
<td>Yes</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>32-bit (sec)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OS Event Timer (OSTimer)</td>
<td>64-bit</td>
<td>Yes</td>
<td>✓</td>
</tr>
</tbody>
</table>

1. $t_{pitc}$ has ±10% accuracy
2. $t_{mrtc}$ has ±3% accuracy
3. $t_{mrc}$ has ±1% accuracy
4. $t_{erc}$ has ± accuracy depends on crystal spec
5. High-frequency crystal that input to main_pll usually is between 10 – 50ppm

### Peripherals to wake from deep sleep

The following peripherals can be used as a wake up source for deep sleep: DMA, GPIOs, Flexcomm (SPI, I²C, I²S, USART), ADC, DMIC, Hypervisor, Secureviolation, HW VAD, RNG, MU, FlexSPI, SDIO, I³C, USB, PUF, PowerQuad, Casper, PMIC, and SHA.
Wake up process

\( T_{\text{amb}} = 25 \, ^\circ\text{C} \), using IRC as the system clock.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from sleep mode, 250 MHz</td>
<td>5.2</td>
<td>1.5</td>
<td>-</td>
<td>( \mu\text{s} )</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from sleep mode, 12 MHz</td>
<td>6.2</td>
<td>-</td>
<td>-</td>
<td>( \mu\text{s} )</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from deep-sleep mode</td>
<td>637</td>
<td>-</td>
<td>-</td>
<td>( \mu\text{s} )</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from deep power-down mode using RESETN</td>
<td>5.6</td>
<td>-</td>
<td>-</td>
<td>ms</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from deep power-down mode using PMIC_IRQ_N.</td>
<td>7</td>
<td>-</td>
<td>-</td>
<td>ms</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from full deep power-down mode using RESETN</td>
<td>5.6</td>
<td>-</td>
<td>-</td>
<td>ms</td>
</tr>
<tr>
<td>( t_{\text{wake}} )</td>
<td>wake-up time</td>
<td>from full deep power-down mode using PMIC_IRQ_N.</td>
<td>7.6</td>
<td>-</td>
<td>-</td>
<td>ms</td>
</tr>
</tbody>
</table>

[1] Typical ratings are not guaranteed. The values listed are at room temperature (25 \(^\circ\text{C}\)), nominal supply voltages.

[2] The wake-up time measured is the time between when a GPIO input pin is triggered to wake the device up from the low power modes and from when a GPIO output pin is set in the interrupt service routine (ISR) wake-up handler.

[3] IRC disabled, all peripherals off, PLL disabled.

[4] Wake up from deep power-down causes the part to go through entire reset process. The wake-up time measured is the time between when the Wake-Up pin is triggered to wake the device up and when a GPIO output pin is set in the reset handler.

Figure 4 Wake up time from low power modes

RAM partitions and power consumption

- Running advanced-while1 in different partition (2 to 29)
- Typical, 1.1V, 25, NBB, 198 MHz (PLL)
- All SRAM array/periphery are ON (set in PDRUNCFG2 and PDRUNCFG3) and one partition clocked using AHB_SRAM_ACCESS_DISABLE register.

![Graph showing VDD1V8_DRV Current (mA) vs Partition # for different data points and partition sizes.](image-url)
Core operation in Active Mode

Maximum core operation frequencies and minimum core voltage in Active Mode

Some power consumption can be achieved by lowering the core voltage and core frequency. Below are the tables showing the maximum operation frequencies and minimum voltage. The CPU frequencies shown below must have SYSCPUAHBCCLKDIV set to 0 (divided by 1).

<table>
<thead>
<tr>
<th>Maximum Core Frequency (-20 °C to +70 °C)</th>
<th>Maximum Core Frequency (0 °C to +70 °C)</th>
<th>Minimum VDD Core Voltage</th>
</tr>
</thead>
<tbody>
<tr>
<td>65 MHz</td>
<td>75 MHz</td>
<td>0.7V</td>
</tr>
<tr>
<td>140 MHz</td>
<td>155 MHz</td>
<td>0.8V</td>
</tr>
<tr>
<td>210 MHz</td>
<td>220 MHz</td>
<td>0.9V</td>
</tr>
<tr>
<td>270 MHz</td>
<td>275 MHz</td>
<td>1.0V</td>
</tr>
<tr>
<td>300 MHz</td>
<td>300 MHz</td>
<td>1.13V</td>
</tr>
</tbody>
</table>

Body Bias

CMOS transistors usually have three terminals – a source, gate and drain, it is also common to have a fourth terminal to the body or substrate. The threshold voltage can be affected by voltage source and voltage body.

RT685 has three types of body bias: Forward Body Bias (FBB), Reverse Body Bias (RBB) and Normal Body Bias (NBB).

<table>
<thead>
<tr>
<th>Body Bias</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBB</td>
<td>Lower the threshold voltage and allows the device to turn on quickly for high performance, but has higher current leakage</td>
</tr>
<tr>
<td>RBB</td>
<td>Increase the threshold voltage and allows the device to turn on slowly, and it reduces current leakage</td>
</tr>
<tr>
<td>NBB</td>
<td>Source voltage and body voltage are matching</td>
</tr>
</tbody>
</table>
Power Lab 1 – Cortex M33 Operating frequency at 0.7V VDDCore
MCUXpresso SDK power_manager example

1. Open the MCUXpresso IDE 11.1.1 or later.

![MCUXpresso IDE](image1)

*Figure 5 MCUXpresso IDE*

2. Creating MCUXpresso new workspace.

![workspace](image2)

*Figure 6 workspace*

Create a project name and click **Launch** to continue.

3. Skip this step if SDK 2.7.0 and above is installed.
   Drag and drop downloaded MCUXpresso SDK to **Installed SDKs** panel at the bottom of MCUXpresso IDE.

![Installed SDKs](image3)

*Figure 7 Installed SDK*

4. Import project.
   Select and click **Import project(s) from file system** link at the bottom left corner panel of IDE.
5. Import project from file system.
   Click ![Browse](#) to select `power_manager.zip` project. Select Next to go to next step.

6. Select project to complete the project import.
   Make sure ![Exminster685_power_manager](#) is selected, then press Finish to complete the import.
PCA9420 and powerlib API

7. SW1_OUT of PCA9420 and VDDCore

VDDCore voltage is connected to SW1_OUT of PCA9420. Current SW1_OUT is programmed to output at 1.00V.
8. Understanding PCA9420 MODECFG register and using powerlib API

There are 6-bit settings for SW1_OUT in MODECFG_0_0 registers (MODECFG_0_0, address 22h).

### Table 5 PCA9420 MODECFG register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Symbol</th>
<th>Default Value</th>
<th>Type</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>SW1_DN</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>ON_MODE</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>SW1_OUT</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The `BOARD_SetPmicVoltageForFreq()` power API in pmic_support.c setups desired VDDCore voltage without user provides value in above table. Simply passing Core and DSP frequency (Table 2), the power API will assign appropriate voltage for the VDDCore.

```c
void BOARD_SetPmicVoltageForFreq(power_part_temp_range temp_range, uint32_t main_clk_freq, uint32_t dsp_main_clk_freq)
```

```c
temp_range = { kPartTemp_0C_P70C = 0U or kPartTemp_N20C_P70C = 1U } 
main_clk_freq or dsp_main_clk_freq range up to 600,000,000
```

(NOTE: Do NOT use PCA9420_WriteModeConfigs() to write value over 011010b!!! This will cause permanent damage to the core.)

9. PCA9420 comes with voltage comparator on each rail (LDO1_OUT, LDO2_OUT, SW1_OUT and SW2_OUT), it compares the actual output voltage against 90% and 110% of its target value. When power-good is enabled, the voltage comparator reports “power-good” status if the nominal voltage is within 90% to 110%. When power-good is disabled, the nominal voltage is set outside the range of 90% to 110%.

Current power_manager example has power-good disabled, when calling `BOARD_SetPmicVoltageForFreq()` to set VDDCore at 0.70 V. Actual measurement on VDDCore is approx. 0.74 V. Change the `kPCA9420_PgoodDisabled` to `kPCA9420_PgoodEnabled` in `BOARD_InitPmic()`. This will put the VDDCore close to 0.70 V.
Core Frequency

10. Changing Core frequency

Before calling `BOARD_BootClockRUN()`, the default 48 MHz (FFRO or 48/60m IrC) CPU frequency is derived from 40/60m IrC (MAINCLKSELA = 3, MAINCLKSELB = 0, and SYSCPUAHBCLKDIV = 0, see green highlight in Figure 12).

Code executes after `BOARD_BootClockRUN()` sets the CPU to run at 250 MHz from main_pll_clk (MAINCLKSELB = 2, and SYSCPUAHBCLKDIV = 1, see orange highlight in Figure 12). The current value from VCO to Main PLL is 528 MHz, the PFD output frequency formula is $FPFDn = FSYSPLL\_OUT \times (18 / PFDn)$. In `BOARD_BootClockRUN()`, PFD0 is set to 19. main_clk is $528,000,000 \times (18 / 19) = 500,210,526$ Hz. MAINPLLCLKDIV is 0 (divided by 1). However, SYSCPUAHBCLKDIV is divided by 2; therefore, the CPU clock is $500,210,526 / 2 = 250,105,263$ Hz.

![Figure 12 CPU Clock Source Overview](image12)

We will be focused more on the clock setup at main_clk (after MAINCLKSELA and MAINCLKSELB). To setup CPU to run less than 65 MHz, the closer frequency it can output is 60 MHz from 48/60m IrC.

If the CPU clock source from the same clock source tries to change, the CPU clock must switch to another clock source before lowering the CPU clock. For example,
the CPU is now using FFRO 48 MHz as clock source, and will be changed to FFRO 60 MHz. The CPU clock must be on another clock source such as SFRO or PLL before the change can be taken place.

In this lab example, the FFRO is set to run at 60 MHz from 48 MHz after the CPU is set to use the clock source from main_clk, then change the CPU clock source to 16m_irc (SFRO see blue highlight in Figure 12). The CPU frequency is now running at 8 MHz, the SYSCPUAHBCCLKDIV is still divided by 2. The flash clock needs to adjust not to exceed core frequency. The flash is using main_pll_clk, so 500 MHz divided by 5 will generate 100 MHz flash clock. Switch the flash clock to FFRO with divided by 10 before switch the CPU clock source, this put the flash clock at 6 MHz. Set the SYSCPUAHBCCLKDIV to 0 (divided by 1), the CPU frequency is at 16 MHz. Call BOARD_SetPmicVoltageForFreq() to set VDDCore to 0.70 V. Attach CPU clock source from SFRO to FFRO, the CPU is now running at 60 MHz. Adjust the flash clock again to run from FFRO at 30 MHz.

```
CLOCK_EnableFfroClk(kCLOCK_Ffro60M);
BOARD_SetFlexspiClock(3U, 10U);
CLOCK_AttachClk(kSFRO_to_MAIN_CLK); /* Let CPU run on ffro before configure SYS PLL. */
CLOCK_SetClkDiv(kCLOCK_DivSysCpuAhbClk, 1U); /* Set SYSCPUAHBCCLKDIV divider to value 1 */
BOARD_SetPmicVoltageForFreq(kPartTemp_BC_P70C, 60000000, 0);
CLOCK_AttachClk(kFFRO_to_MAIN_CLK); /* Let CPU run on ffro before configure SYS PLL. */
BOARD_SetFlexspiClock(3U, 2U);
SystemCoreClock = CLOCK_GetFreq(kCLOCK_CoreSysClk);
```

When using internal LDO, replace BOARD_SetPmicVoltageForFreq() with POWER_SetLdoVoltageForFreq(), the parameters are the same as PMIC. Remove JP29 and change JP22 to 1-2 as shown in Figure 13 VDDCore and LDO_Enable location Figure 13.

NOTE: JP12 1-2 MUST be shunt when using internal LDO, make sure SW2_OUT 1.8V is supplied to VDDIO1 for the internal core voltage. DO NOT use internal LDO if JP12 2-3 is shunt.

MCUXpresso SDK power_manager example
11. VDDCore and LDO_ENABLE location

Take an unused jumper as indicated in the image below and place in on 2-3 of JP22. Place multimeter to JP29 to measure VDDCore or place amp meter to JP29 (remove jumper) to
measure current.

**Figure 13 VDDCore and LDO_Enable location**

**NOTE:** JP12 1-2 **MUST** be shunt when using internal LDO, make sure SW2_OUT 1.8V is supplied to VDDIO1 for the internal core voltage. **DO NOT** use internal LDO if JP12 2-3 is shunt.

**Figure 14 LDO_ENABLE selection**

**Figure 15 JP12 VDDIO1 power source selection**

12. Power the board
   Connects USB micro cable from PC to J5 (LPCLink as indicated in Figure 13).

**Serial Terminal**
13. Open Serial Terminal program such as TeraTerm as shown below.
   Select serial port listed as Jlink CDC UART Port, and press to **OK** continue.
Setup the serial port settings, and press **OK** to continue.

**Figure 17 Serial port setup**

**Build the project**
14. Build the example
   
   Under the Quickstart Panel at the bottom left, click the **Build** under the **Build your project**.

**Figure 18 Quickstart Panel - Build**

At the middle bottom of IDE, the console will show the project built successfully.
Run the project
15. Program the application

Make sure the SW5 (ISPs) is ON, OFF, ON from left to right.
Under the Quickstart Panel at the bottom left, click the Debug under the Debug your project.

16. A window will pop up to select debugger connects to the EVK. Select I-Link LPCXpresso V2 and press Ok to continue.
When it is successfully programmed, the program count will be halted at main() as shown in the source window.

You can continue to run the program by pressing (resume). Or terminate the debug program by pressing , then press the reset button on the board (see Figure 13).

17. Either continue to run the program from debug or from reset, message will be shown as below.
18. Press 5 on the Terminal to change the VDDCore to 0.70 V, or press 6 to change the VDDCore to 1.15 V. Place a multimeter on JP29 to measure the voltage change when pressing number 5 or 6 key on the serial terminal.
Power Lab 2 – Wake from deep sleep using RTC, Pin Interrupt and USART

RTC wake (high resolution counter) as wake up source

1. Initialize RTC module and enable the RTC wake in STARTEN
   - Enable the RTC output of the RTC oscillator 32,768 Hz. Failure to do so, the RTC counter will not function.

   After initialize the RTC by calling `RTC_Init()`, calls `RTC_StartTimer()` to enable the RTC. There are two type of wake-up features in RTC: WAKE and ALARM. WAKE is using 1 khz counter (milli seconds) up to 65.5s (0xFFFF) and ALARM is using RTC counter (seconds) up to 100+ years (0xFFFF_FFFF). WAKE interrupts are enabled in RTC by calling `RTC_EnableInterrupts()`. STARTEN and IRQ will be set when calling `EnableDeepSleepIRQ()`.

```c
int main(void) {
    CLKCTL0->OSC32KHZCTL0 = 1;
    /* Initialize RTC timer */
    RTC_Init(RTC);
    RTC_StartTimer(RTC);
    /* enable RTC interrupt */
    RTC_EnableInterrupts(RTC, RTC_CTRL_WAKEDPD_EN_MASK);
    /* Enable RTC wake */
    EnableDeepSleepIRQ(RTC_IRQn);
    EnableIRQ(RTC_IRQn);
    BOARD_InitPins();
    BOARD_BootClockRUN();
}
```

2. RTC IRQ Handle
   - It is essential to have RTC IRQ to clear the flag after woke up from deep sleep. The flag needs to be cleared in order to avoid deadlock.

```c
void RTC_IRQHandler(void) {
    if (RTC_GetStatusFlags(RTC) & kRTC_WakeupFlag) {
        /* Clear wake flag */
        RTC_ClearStatusFlags(RTC, kRTC_WakeupFlag);
    }
    if (RTC_GetStatusFlags(RTC) & kRTC_AlarmFlag) {
        /* Clear alarm flag */
        RTC_ClearStatusFlags(RTC, kRTC_AlarmFlag);
    }
}
```
3. Provides a counter value to RTC to wake from deep sleep. A write to this register will immediately start count down sequence in ms. In the example, 10000 (ms) is passed to RTC_SetWakeupCount(), the RT685 will wake from deep sleep after 10s.

```c
case kPmu_Deep_Sleep: /* Enter deep sleep mode. */
  #if Defined POWER_DOWN_PLL_BEFORE_DEEP_SLEEP
    /* Disable PLL before enter deep sleep mode */
    BOARD_DisablePll();
  #endif
  RTC_SetWakeupCount(RTC, 10000);
  POWER_EnterDeepSleep(APP_EXCLUDE_FROM_DEEPSLEEP);
  #if Defined POWER_DOWN_PLL_BEFORE_DEEP_SLEEP
    /* Restore PLL before enter deep sleep mode */
    BOARD_RestorePll();
  #endif
  break;
```

Note: Other timers that described in Timers function during deep sleep and deep power down use similar approach – setup timer IRQ Handle, setup timer clock source with the clock source excludes from deep sleep, set STARTEN and interrupt for the timer, and set time out on the timer before entering deep sleep.

```c
static void UTickCallback(void)
{
}

CLOCK_AttachClk(hLPOS_C_to_UTICK_CLK);
UTICK_Init(UTICK0);
UTICK_SetTick(UTICK0, kUTICK_Onetime, 1000000UL - 1, UTickCallback);
EnableDeepSleepIRQ(UTICK0_IRQHandler);
POWER_EnterDeepSleep(APP_EXCLUDE_FROM_DEEPSLEEP);
```

Pin Interrupt as wake up source

GPIO pins can be used as wake up source for deep sleep. Two type of interrupts for GPIO – Pin Interrupt (PIINT) or GPIO interrupt. There are a total of 8 Pin Interrupts and 2 GPIO group interrupts in RT685. Current example is using Pin interrupt. During the deep sleep, user can press SW1 user button to wake from deep sleep.

```c
static void APP_InitWakeUpPin(void)
```
USART as another wake up source
There are two methods in deep sleep mode for USART. One is using 32 khz (RTC or lp_32k) runs in async mode, and another is sync mode that requires additional pin for clock to provide from master.

In this example, before RT685 goes to deep sleep, enabled the 32 khz clock source in USARTn->CFG. The WAKECLK32KHZSEL selection is assigned USART to use either RTC 32 khz or lp_32k (derived from 1m_osc then divided by 32, 1m_osc must be excluded from deep sleep).

This example uses RTC since it is turned on and its accuracy is better than lp_32k. Next step is to store the BRG and OSR value for baudrate restore before switching to 9600 baudrate. Enable USART Rx Interrupt(s), STARTEN and Flexcomm IRQ before going deep sleep.

Any data receive from USART Rx will wake up from deep sleep. Disable the 32k clock and quickly turn off the IRQ to prevent next Rx data to trigger interrupt since the current console output does not use interrupt for serial console. Restore the baud rate from 9600 to 115200.

```c
void FLEXCOMM0_IRQHandler(void)
{
    uint8_t data;
    /* If new data arrived. */
    if (((USART_RxFifoNotEmptyFlag | kUSART_RxError) & USART_GetStatusFlags(USART0))
    {
        data = USART_ReadByte(USART0);
    }
}```
Understanding deep sleep parameters that passes to deep sleep power API
The power API to enter deep sleep is `POWER_EnterDeepSleep()`, the API passes value of array.
Parameter 0: RUN/SLEEPCFG
Parameter 1: Peripherals RAM
Parameter 2: Array RAM partitions
Parameter 3: Periphery RAM partitions

When the bit is set in the parameter, it will be excluded from deep sleep. For example, FFRO needs to be active during deep sleep, set bit 16 of parameter 0. (SYSCTL0->PDSLEEPCFG0 in User Manual).
In the example, parameter 1 has FLEXSPI RAM excluded, parameter 2 has 21 array partitions excluded.

```c
#define APP_DEEPSLEEP_RUNCFG0 (SYSCTL0_PDSLEEPCFG0_RBB_PD_MASK | SYSCTL0_PDSLEEPCFG0_FFRO_PD_MASK)
#define APP_DEEPSLEEP_RAM_APD 0xFFFFF8U
#define APP_DEEPSLEEP_RAM_PPD 0x0U
#define APP_EXCLUDE_FROM_DEEPSLEEP
  (((const uint32_t[]){APP_DEEPSLEEP_RUNCFG0, 
    (SYSCTL0_PDSLEEPCFG1_FLEXSPI_SRAM_APD_MASK | SYSCTL0_PDSLEEPCFG1_FLEXSPI_SRAM_PPD_MASK), 
    APP_DEEPSLEEP_RAM_APD, APP_DEEPSLEEP_RAM_PPD}))
```

Build the project
4. Build the example
   Under the Quickstart Panel at the bottom left, click the Build under the
   Build your project.

   ![Figure 24 Quickstart Panel - Build](image)

   At the middle bottom of IDE, the console will show the project built successfully.

   ![Figure 25 CDT Build Console](image)

Run the project
5. Program the application
Make sure the SW5 (ISPs) is ON, OFF, ON from left to right.

Under the Quickstart Panel at the bottom left, click the Debug under the "Debug your project".

![Quickstart Panel - Debug](image)

6. A window will pop up to select debugger connects to the EVK. Select J-Link LPCXpresso V2 and press OK to continue.

![Selecting debugger](image)

When it is successfully programmed, the program count will be halted at main() in the source window as shown below.
Terminate the debug program by pressing \( \text{[Stop]} \), then press the reset button on the board (see Figure 13).

NOTE: Do not use debugger when the code is going to perform low power modes. The debugger will shut off during low power modes, this will cause the IDE not response.

Deep sleep wake by RTC
7. After reset, message will be shown as below. Press 2 on the serial terminal window to enter deep sleep.

8. After wake has elapsed, it will back to the selections.
Deep sleep wake by Pin

9. Press 2 again to enter deep sleep, this time using SW1 User button to wake from deep sleep prior to RTC wake (10s).

10. After wake, it will back to the selections.
Deep sleep wake by USART

11. Press 2 again to enter deep sleep, this time enter any key on the serial terminal window to wake from deep sleep prior to RTC wake (10s).

12. After wake, it will back to the selections.

Figure 33 Deep sleep wake by USART
1. Configure the RTC alarm to wake from deep power down. Append
   
   ```c
   RTC_CTRL_ALARMDPD_EN_MASK
   ```
   to `RTC_EnableInterrupts()`. 

   ```c
   RTC_StartTimer(RTC);
   /* enable RTC interrupt */
   RTC_EnableInterrupts(RTC, RTC_CTRL_WAKEDPD_EN_MASK | RTC_CTRL_ALARMDPD_EN_MASK);
   /* Enable RTC wake */
   EnableDeepSleepIRQ(RTC_IRQHandler);
   ```

2. Setup RTC time and alarm
   
   Setup the date and convert to seconds using `RTC_SetDatetime()`. Add 20s to the
date.second and pass it to `RTC_SetAlarm()`. Or, direct register write using `RTC->MATCH = RTC->COUNT + 20;` 

   ```c
   int main(void)
   {
     /* Init board hardware. */
     pca9420_modecfg_t pca9420ModeCfg[4];
     uint32_t i;
     rtc_datetime_t date;

     ...
   
     case kPmu_Deep_PowerDown: /* Enter deep power down mode. */
       PRINTF("Press any key to confirm to enter the deep power down mode and wakeup the
device by ");
       "reset.\r\n\r\n";
       GETCHAR();

       date.year = 2020;
       date.month = 4;
       date.day = 6;
       date.hour = 9;
       date.minute = 0;
       date.second = 0;
       RTC_SetDatetime(RTC, &date);
       date.second += 20;
       RTC_SetAlarm(RTC, &date);
       POWER_EnterDeepPowerDown(APP_EXCLUDE_FROM_DEEP_POWERDOWN);
       break;
   ```

Build the project

3. Build the example

   Under the Quickstart Panel at the bottom left, click the `Build` under the
   Build your project.
At the middle bottom of IDE, the console will show the project built successfully.

Run the project

4. Program the application

Make sure the SW5 (ISPs) is ON, OFF, ON from left to right.

Under the Quickstart Panel at the bottom left, click the **Debug** under the **Debug your project**.

5. A window will pop up to select debugger connects to the EVK. Select **J-Link LPCXpresso V2** and press **OK** to continue.
When it is successfully programmed, the program count will be halted at `main()` in the source window as shown below.

Terminate the debug program by pressing , then press the reset button on the board (see Figure 13).

Deep power down wake by deep sleep
6. After reset, selection message will show. Press 3 in the serial terminal to enter deep power down.
7. Manually reset the flash while in deep power down. Use a female to female jumper wire, one end connects to GND (see Figure 40), one end connects to male to male header.

The reason is when RT685 wakes up by RTC from deep power down, the flash is still in previous mode (Quad or Octal). ROM will not be able to access flash content correctly; therefore, RT685 is not able to execute code from flash.

There is an GPIO pin connects to nRESET_OSPI of the reset pin on the SPI flash. User can configure the P2_12 to be asserted in the OTP. This lab chooses not to use OTP to configure the P2_12; incorrectly programming the OTP will cause unpredictable behavior.
8. Locate TP34 or TP33 near the standard SD slot.

9. Press any key in the serial terminal to confirm to enter the deep power down mode.

10. Quickly use the female-to-female jumper wire with male header to touch TP33 or TP34, and untouch it.

11. When RT685 wake up by RTC and boot up, it will back to the selection.
Deep power down wake by reset

12. Press 3 to enter deep power down.

13. Press any key to confirm in the serial terminal to enter the deep power down mode.

14. Press reset to wake from deep power down.

Measuring power consumption on VDDCore JP29

Different multi-meter has different burden voltage when using the current measurement (mA/A and mA/uA). High end current measurement equipment provides wider range of burden voltage that will not draw current from the VDDCore. Handheld such as Fluke 83 is 0.03V in A, 1.8mV in mA and 100uV in uA, and the resistance between the input terminals is 0.03 ohm for A, 1.8 ohm for mA,
and 100 ohm for uA. Assuming 40mA VDDCore is requirement in active mode, 40mA * 1.8mV = 74mV. The VDDCore is dropped below 1.0V (1.0V – 74mV = 926mV).

The solution is to raise the VDDCore voltage from 1.0V to 1.1V before changing the core frequency from FFRO to PLL. Enable SFRO after BOARD_InitPins() and before BOARD_BootClockRUN(). It is required for PMIC configuration before BOARD_BootClockRUN().

```c
void BOARD_ConfigPMICModes(pca9420_modecfg_t *cfg, uint32_t num) {
    assert(cfg);
    /* Configuration PMIC mode to align with power lib like below:
     *  0b00  run mode, no special.
     *  0b01  deep sleep mode, vddcore 0.7V.
     *  0b10  deep powerdown mode, vddcore off.
     *  0b11  full deep powerdown mode vdd1v8 and vddcore off. */
    cfg[0].sw1OutVolt = kPCA9420_Sw1OutVolt1V100;
    ...
}

int main(void) {
    ...
    BOARD_InitPins();
    /* Configure SFRO clock */
    POWER_DisablePD(kPDRUNCFG_PD_SFRO); /* Power on SFRO (16MHz) */
    CLOCK_EnableSfroClk(); /* Wait until SFRO stable */
    /* PMIC PCA9420 */
    BOARD_InitPmic();
    for (i = 0; i < ARRAY_SIZE(pca9420ModeCfg); i++)
    {
        PCA9420_GetDefaultModeConf(&pca9420ModeCfg[i]);
    }
    BOARD_ConfigPMICModes(pca9420ModeCfg, ARRAY_SIZE(pca9420ModeCfg));
    PCA9420_WriteModeConfigs(&pca9420Handle, kPCA9420_Mode0, &pca9420ModeCfg[0],
ARRAY_SIZE(pca9420ModeCfg));
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    CLOCK_EnableFfroClk(kCLOCK_Ffro60M);
    POWER_SetPadVolRange(&vrange);
    ...
}
```