

**ECEN 5813 Final Project Report**  
**Low Power Mode Demo and Experiments on FRDM**  
**KL25Z**

**Contents**

<b>1. Introduction.....</b>	<b>3</b>
<b>2. Low Power Modes.....</b>	<b>3</b>
<b>3. Building and Running ‘power_mode_switch’ Application from MCU-SDK on MCUXpresso .....</b>	<b>5</b>
<b>3.1. Working .....</b>	<b>5</b>
<b>4. Building and Running Touch Sensing Software (TSS) from NXP .....</b>	<b>8</b>
<b>4.1. Working .....</b>	<b>8</b>
<b>5. Proposed Method to Use Accelerometer as a Wakeup Source .....</b>	<b>9</b>
<b>6. References.....</b>	<b>9</b>

## 1. Introduction

The objective of this project has been to learn about the low-power modes supported by FRDM KL25Z board by running experiments and getting findings based on that.

Following are the two major experiments covered concerning the Low Leakage Stop (LLS) mode in this document:

- The modification in 'power mode switch' MCU-SDK application to make system transition from RUN mode to the Low Leakage Stop (LSS) mode using Touch Sensor Input (TSI).
- How to build and run the TSS library released by the NXP.

## 2. Low Power Modes

The power management controller (PMC) provides multiple power options to allow the user to optimize power consumption for the level of functionality needed. Depending on the stop requirements of the user application, a variety of stop modes are available that provide state retention, partial power down or full power down of certain logic and/or memory. I/O states are held in all modes of operation. The following table compares the various power modes available:[\[1\]](#)

**Table 5. Chip power modes**

Chip mode	Description	Core mode	Normal recovery method
Normal run	Allows maximum performance of chip. Default mode out of reset; on-chip voltage regulator is on.	Run	—
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.	Sleep	Interrupt
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.	Sleep Deep	Interrupt

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; in BLPI clock mode, the fast internal reference oscillator is available to provide a low power nominal 4MHz source for the core with the nominal bus and flash clock required to be <800kHz; alternatively, BLPE clock mode can be used with an external clock or the crystal oscillator providing the clock source.	Run	—
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.	Sleep	Interrupt
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 OSC, LPTMR, RTC, CMP, TSI can be used. TPM and UART can optionally be enabled if their clock source is enabled. 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).	Sleep Deep	Interrupt
LLS (Low Leakage Stop)	State retention power mode. Most peripherals are in state retention mode (with clocks stopped), but OSC, LLWU, LPTMR, RTC, CMP,, TSI can be used. NVIC is disabled; LLWU is used to wake up.  <b>NOTE:</b> 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).	Sleep Deep	Wakeup Interrupt <sup>1</sup>
VLLS3 (Very Low Leakage Stop3)	Most peripherals are disabled (with clocks stopped), but OSC, LLWU, LPTMR, RTC, CMP, TSI 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).	Sleep Deep	Wakeup Reset <sup>2</sup>
VLLS1 (Very Low Leakage Stop1)	Most peripherals are disabled (with clocks stopped), but OSC, LLWU, LPTMR, RTC, CMP, TSI can be used. NVIC is disabled; LLWU is used to wake up.  All of SRAM_U and SRAM_L are powered off.	Sleep Deep	Wakeup Reset <sup>2</sup>
VLLS0 (Very Low Leakage Stop 0)	Most peripherals are disabled (with clocks stopped), but LLWU, LPTMR, RTC, TSI can be used. NVIC is disabled; LLWU is used to wake up.  All of SRAM_U and SRAM_L are powered off.  LPO shut down, optional POR brown-out detection	Sleep Deep	Wakeup Reset <sup>2</sup>

1. Resumes normal run mode operation by executing the LLWU interrupt service routine.
2. Follows the reset flow with the LLWU interrupt flag set for the NVIC.


### 3. Building and Running the modified ‘power\_mode\_switch’ Application from MCU-SDK on MCUXpresso

Following are the steps to be followed to build the ‘power\_mode\_switch’ application modified and pushed on the [github](#):

- Open a workspace
- *File->Import->General->Existing Projects into Workspace-><select the [github](#) project>*
- Build the project
- *Quick Settings->SDK Debug Console->UART Console*
- *Debug your project->Program flash action using PEMicro probes->Select the PEMicro probe*
- Open a serial terminal and run the demo

#### Running the demo:

- The system is in the RUN mode in the beginning. So, you should be able to see the green LED on the board.
- Whenever a touch is detected on the touch sensor, the system transitions to the LLS mode by blinking the LED for some time and then turning it off.
- The system will wake up automatically in 8 seconds and the LED will turn back on.
- The following screenshot shows the debug console output for the demo:



```
The system is going into LLS mode.
Will be woken up by LPTMR in 8 seconds
```

#### 3.1. Working

In the LLS mode, the triggering source for LLS mode is set as touch sensor input and wakeup source is set as Low Power Timer (LPTMR). I have configured the LPTMR for 8 seconds.

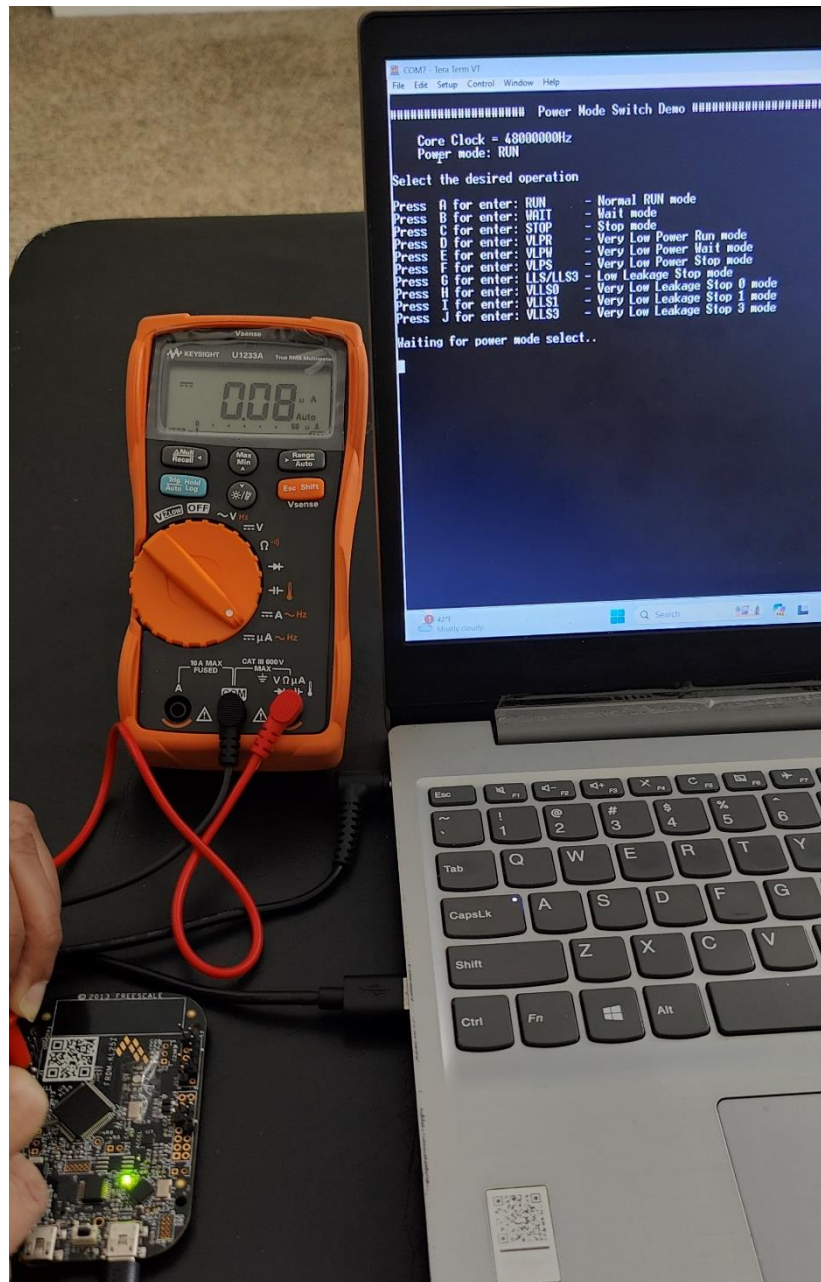
As as the internal module 0 is dedicated for LPTMR, The LPTMR is specified as wakeup source in the Low-Leakage Wakeup Unit (LLWU) by writing 1 to its Module Enable (ME) register as follows:

*moduleIndex = 0;*

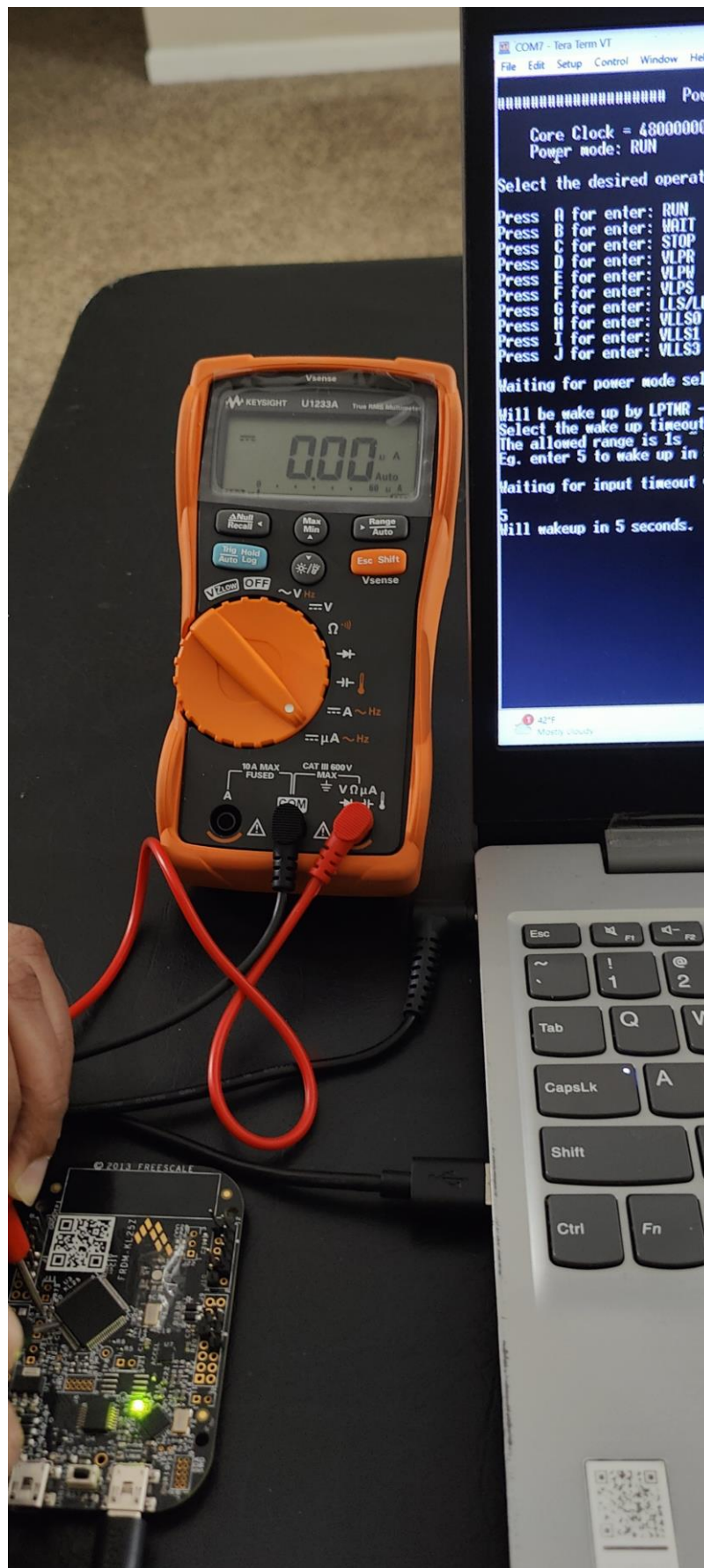
***Fsl\_llwu.h: LLWU\_EnableInternalModuleInterruptWakeup:*** *base->ME |= 1U << moduleIndex;*

Before entering into LLS mode, the debug console is disabled from ‘APP\_PowerPreSwitchHook’ and re-enabled from ‘APP\_PowerPostSwitchHook’ API. That means whenever the system enters into LLS mode, the debug console will be unresponsive and will again be responsive when it transitions back to the RUN mode.

The current drawn can be measured by connecting one probe of ammeter to the R81 of J4 and the 7<sup>th</sup> pin of MKL25Z128VLK4 chip (P3V3\_KL25Z) on the board as follows:

**RUN Mode:**



**LLS Mode:**

## 4. Building and Running Touch Sensing Software (TSS) from NXP

TSS can be downloaded from <https://www.nxp.com/design/design-center/sensor-developer-resources/touch-sensing-software:TSS?SAMLart=ST-AAF0E3YJDJej%2BJVBprc7Vu5rkUdez%2FVIRhj1SaMxc09M%2FjBq1sXEckki>.

Once downloaded, follow the setup steps. They are self-explanatory.

For KXX boards, The TSS library is supported for CodeWarrior, IAR Embedded Workbench and MDK-ARM uVision IDEs. The TSS library was verified with the IAR Embedded Workbench IDE – Arm 9.50.1 by following steps:

- *Project->Add Existing Project...->Select the*  
‘Freescale\_TSS\_3\_1\examples\FRDMKLXX\_DEMO\build\iar\FRDMKL25Z\_DEMO.ewp’.
- *Project->Make*
- *Project->Download and Debug*
  - Note: There was an error for FlashKLxx128K.board. To resolve it, it was required to rename arm\config\flashloader\NXP\ to **Freescale**.
- Run the demo

The demo video is pushed on [Demo video](#).

### 4.1. Working

In the FRDMKL25Z\_DEMO demo, low-power function is added; if no touch is detected in about 8 seconds, the system will enter low-power mode LOW\_POWER\_MODE. The CPU will wake up if a touch is detected because the TSI out-of-range interrupt can wake the MCU in all kinds of low-power modes.

There is an issue in the current KL25 chips that the out-of-range interrupt function is closely connected with the end of scan interrupt. The user can not wake the CPU with the out-of-range interrupt independently, so the demo gives out a workaround. The LPTMR is used as the LLWU source too. Every time the LPTMR interrupt occurs, the TSI0\_GENCS register will be polled and the End of Scan flag is cleared, or the scan of electrode will not be executed.<sup>[2]</sup>

Following are the LLWU configurations:

*/\* Setup LLWU (Low Leakage wake-up unit) for TSI \*/*

```
psLLWUStruct->ME |= LLWU_ME_WUME4_MASK;    /* TSI LLWU Input Enable =
LLWU_M4IF) */
```

```
psLLWUStruct->ME |= LLWU_ME_WUME0_MASK;    /* LPTMR LLWU Input Enable =
LLWU_M0IF) */
```

The library uses periodic wakeup with the LLS mode to be able to detect the touch inputs. So, whenever the touch is detected, the system goes back to RUN state.

A white LED is used to denote the RUN mode and a blinking green LED for periodic wakeup.

**Note:** I tried porting this library to MCUXpresso to use touch sensor interrupt as a wakeup source but I am unable to get the periodic wakeup completely working.



## 5. Proposed Method to Use Accelerometer as a Wakeup Source

There is no dedicated pin of LLWU connected to the accelerometer interrupt pin.

Reference for similar implementation can be found at <https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/utlising-the-lls-mode-and-the-llwu-for-the-kl25z-series>.

To use accelerometer interrupt as a wakeup source from low power mode, one of the following methods can be implemented:

- Configure the accelerometer interrupt pin as a wakeup pin and connect it to one of the LLWU pins after cutting the existing trace.
- Use two FRDM KL25Z boards.
  - On the first board, run the accelerometer and write the GPIO if the interrupt is received.
  - Connect the GPIO pin to one of the LLWU pins of the second board and run the application on that board.
- **Note:** Above mentioned are just the proposed methods as cutting and connecting the traces on the board was not feasible because of its tiny size.

## 6. References

1. <https://www.nxp.com/docs/en/product-brief/KL25PB.pdf>
2. <https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d6cf956326418558a6a5345df3c7946b350150e>