




**MSP430® FRAM Utilities version 01.00.00.24**

# **USER'S GUIDE**

---

# Copyright

Copyright © Texas Instruments Incorporated. All rights reserved.

 Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this document.

Texas Instruments  
Post Office Box 655303  
Dallas, TX 75265  
<http://www.ti.com/msp430>



## Revision Information

This is version 01.00.00.24 of this document, last updated on April 08, 2015.

---

# Table of Contents

<b>Copyright</b> .....	<b>2</b>
<b>Revision Information</b> .....	<b>2</b>
<b>1 Introduction</b> .....	<b>5</b>
<b>2 Compute Through Power Loss (CTPL)</b> .....	<b>7</b>
2.1 Introduction .....	7
2.2 Usage .....	8
2.2.1 Components .....	8
2.2.2 Code Composer Studio (CCS) .....	10
2.2.3 IAR Embedded Workbench .....	12
2.3 API Reference .....	15
2.3.1 API Overview .....	15
2.3.2 Core API Set .....	15
2.3.3 Low Level .....	21
2.3.4 Peripherals .....	23
2.3.5 Benchmark .....	25
2.4 Examples .....	27
2.4.1 Examples Overview .....	27
2.4.2 LPM4.5 With GPIO Wakeup .....	28
2.4.3 LPM3.5 With RTC Wakeup .....	29
2.4.4 COMP_E Powerloss Monitor .....	30
2.4.5 ADC12_B Powerloss Monitor .....	31
2.5 Benchmarking .....	32
2.5.1 Overview .....	32
2.5.2 Configuration .....	34
<b>IMPORTANT NOTICE</b> .....	<b>36</b>



# 1 Introduction

The Texas Instruments® FRAM Utilities is a collection of embedded software utilities that leverage the ultra-low power and virtually unlimited write endurance of FRAM. The utilities are available for MSP430FRx FRAM microcontrollers and provide example code to help start application development.

Included are the following FRAM Utilities:

- **Compute Through Power Loss:** A utility API set that enables ease of use with LPMx.5 low-power modes and a powerful shutdown mode that allows an application to save and restore critical system components when a power loss is detected.



## 2 Compute Through Power Loss (CTPL)

Introduction .....	7
Usage .....	7
API Reference .....	15
Examples .....	27
Benchmarking .....	32

### 2.1 Introduction

Compute Through Power Loss (CTPL) is a utility API set that leverages FRAM to enable ease of use with LPMx.5 low-power modes and provides a powerful shutdown mode that allows an application to save and restore critical system components when a power loss is detected.

Traditional use of the LPM3.5 and LPM4.5 modes cause the application to reset when waking up and both application and peripheral state are not retained. The application must check for the LPMx.5 reset source at the start of the program and execute a separate branch of code if the device is waking up from a LPMx.5 mode. This often includes reinitializing both core system and application required peripherals in addition to initialization of global variables by the compiler defined c-start up function that is executed before the main program. This increases the start up time and increases the complexity of applications. As a result application programmers often avoid these low-power modes unless absolutely necessary.

The CTPL utility provides an easier solution for the application programmer. The included linker configuration files will place all application data sections into FRAM where they are retained through LPMx.5 low-power modes. The utility will also allocate FRAM storage used to save the state of the application and critical system peripherals. When entering into low-power modes with the CTPL utility the FRAM storage will be used to save the necessary components and the utility will put the device into the specific low power mode and wait for a device wakeup or reset. Upon device wakeup or reset the utility will intercept the reset and restore the application and peripheral state from the FRAM storage. After restoring the state the utility returns back to the application and the next line of code is executed, removing the need for the application programmer to check for a reset at the start of main.

Application execution using LPMx.5 modes and the CTPL utility can now be written using the same methods as LPM0-3 where the system state is retained. This enables existing applications to easily integrate the CTPL utility and begin using LPMx.5 modes in place of existing LPM0-3 modes and avoid rewriting complex application start up code.

Additionally the CTPL utility provides an API to safely save and restore context in the event of a powerloss. The utility will save the state of the application and critical system peripherals just like the LPMx.5 modes and then wait for the device to enter a BOR due to powerloss. A configurable parameter allows for a timeout for situations where the voltage ramps back up to operational levels. A device reset, power on or timeout will all restore the saved state and return to the application in the same manner as the LPMx.5 functions. See the [CTPL examples](#) section for powerloss monitor examples using an internal ADC12\_B window comparator solution and an external COMP\_E solution using a simple voltage divider to detect when power is lost.

## 2.2 Usage

Components .....	8
Code Composer Studio (CCS) .....	10
IAR Embedded Workbench .....	12

### 2.2.1 Components

The CTPL utility consists of the following software components. Some of these are intended to be directly called from the application while others are internal to the utility implementation.

#### 2.2.1.1 Core API Set

The CTPL Core API Set represents utility API's that the application can directly interface with. The simple API set includes the following functions:

- `ctpl_init()`: Initialize the CTPL library at the start of the system pre-init.
- `ctpl_enterLpm35()`: Save context, enter LPM3.5, restore context and return to the main application.
- `ctpl_enterLpm45()`: Save context, enter LPM4.5, restore context and return to the main application.
- `ctpl_enterShutdown()`: Save context, disable all interrupt sources, configure watchdog timeout and wait for BOR. Restore context and return to the main application on a device reset, power on or timeout.

See the [Core API reference](#) for complete API documentation.

#### 2.2.1.2 Low Level

Low-level C and assembly functions that directly interface with the MSP430 to save the state and enter low power modes. These functions are called by the [Core API Set](#) and should not be invoked from the main application.

See the [Low Level reference](#) section for complete API documentation.

#### 2.2.1.3 Peripheral

Peripheral specific functions to save and restore context. Each peripheral supported by the utility has a save, restore and epilogue function that can be defined by the CTPL device file based on peripheral availability and called by the [Core API Set](#).

The CTPL utility currently supports the following peripherals:

- System Resets, Interrupts, and Operating Modes, System Control Module (SYS)
- Power Management Module (PMM)
- Clock System (CS)



- 32-Bit Hardware Multiplier (MPY32)
- FRAM Controller (FRCTL)
- Memory Protection Unit (MPU)
- RAM Controller (RAMCTL)
- Digital I/O (PORT, PORT\_INT)
- Watchdog Timer (WDT\_A)
- Real-Time Clock B (RTC\_B)
- Real-Time Clock C (RTC\_C)

See the [Peripheral reference](#) section for complete API documentation.

#### 2.2.1.4 Device

Device specific C and linker configuration files. Every CTPL application needs to include the device C file that corresponds to the device being used. This device C file defines the peripherals that are saved and restored by the utility. Generally the LPMx.5 device wakeup time is significantly long enough that the peripheral restore routine has minimal impact on the overall wakeup time of the application, however certain peripherals can be excluded if they are not used in the application by editing this device C file. Additionally any CTPL application is required to use the device and IDE specific linker configuration file which places all read/write data into FRAM. Both of these files are included by default in the empty and example projects provided with the utility.

See the [Code Composer Studio \(CCS\)](#) or [IAR Embedded Workbench](#) section for IDE specific instruction on using the CTPL utility.

## 2.2.2 Code Composer Studio (CCS)

### 2.2.2.1 Creating an Empty CTPL Project

FRAM Utilities is a discoverable package in Code Composer Studio (CCS). Creating a new project with the complete CTPL library configured is as easy as selecting the "File -> New -> CCS Project" menu option and selecting the "Empty Project with FRAM Utilities" project template.

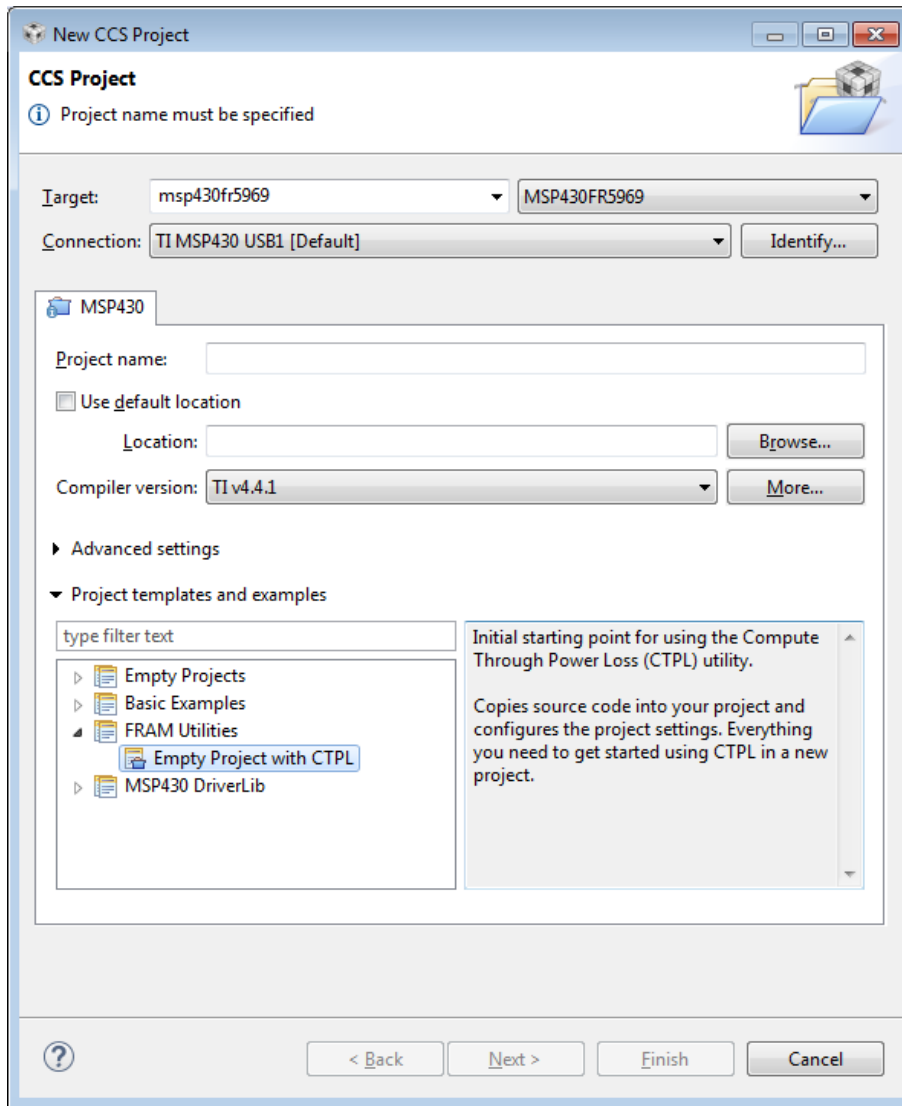


Figure 2.1: CCS new project wizard

### 2.2.2.2 Add CTPL to an Existing Application

The same project template can be used to apply the FRAM Utilities and CTPL settings and source code to an existing CCS project. Right click the project and select the "Source -> Apply Project

Template..." menu option and select the "Add Copy of FRAM Utilities to Project" project template.

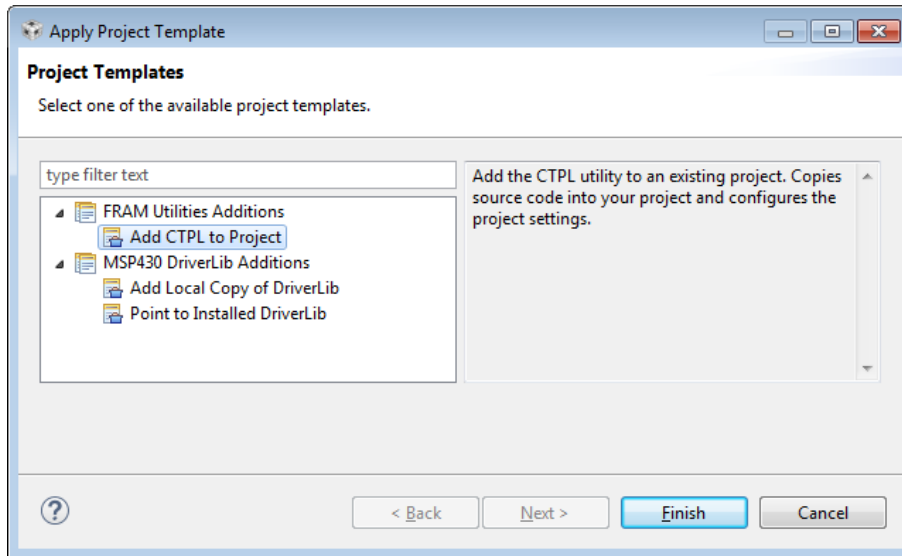


Figure 2.2: CCS apply project template

## 2.2.3 IAR Embedded Workbench

### 2.2.3.1 Opening the Examples

The CTPL utility provides an IAR Embedded Workbench workspace with preconfigured projects for each example. The workspace can be opened in IAR Embedded Workbench by double clicking the .eww file if windows associates this file type with IAR Embedded Workbench. Alternatively the workspace can be opened within IAR Embedded Workbench by navigating to and selecting the desired workspace in the "File -> Open -> Workspace" menu option.

### 2.2.3.2 Add CTPL to an Existing Application

Using the CTPL utility with IAR Embedded Workbench requires several different step to configure properly. The steps have been listed below and need to be followed closely to ensure proper integration with the existing application.

1. Add the CTPL source code to the project.
2. Add the CTPL include path to the project compiler options.

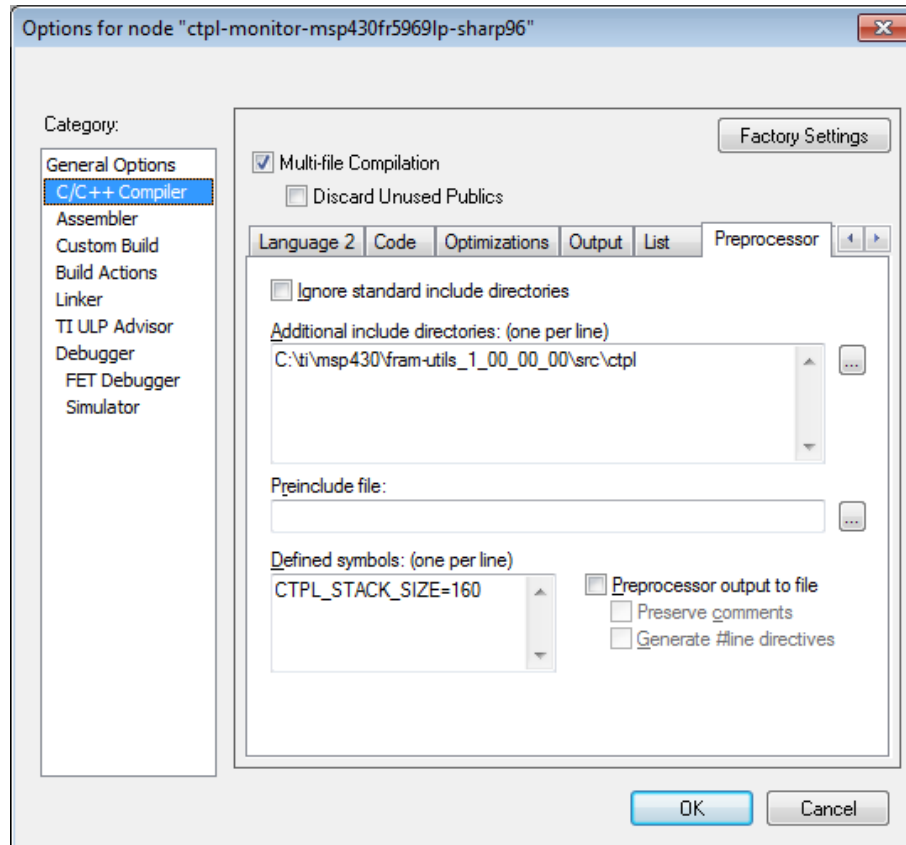


Figure 2.3: CTPL include path

3. Add the required predefined assembler symbols to the project assembler options.

- (a) CTPL\_STACK\_SIZE is required and must be predefined to the same size as the configured stack size.
- (b) \_\_LARGE\_CODE\_MODEL\_\_ is optional and should only be predefined if the project uses the large code model.

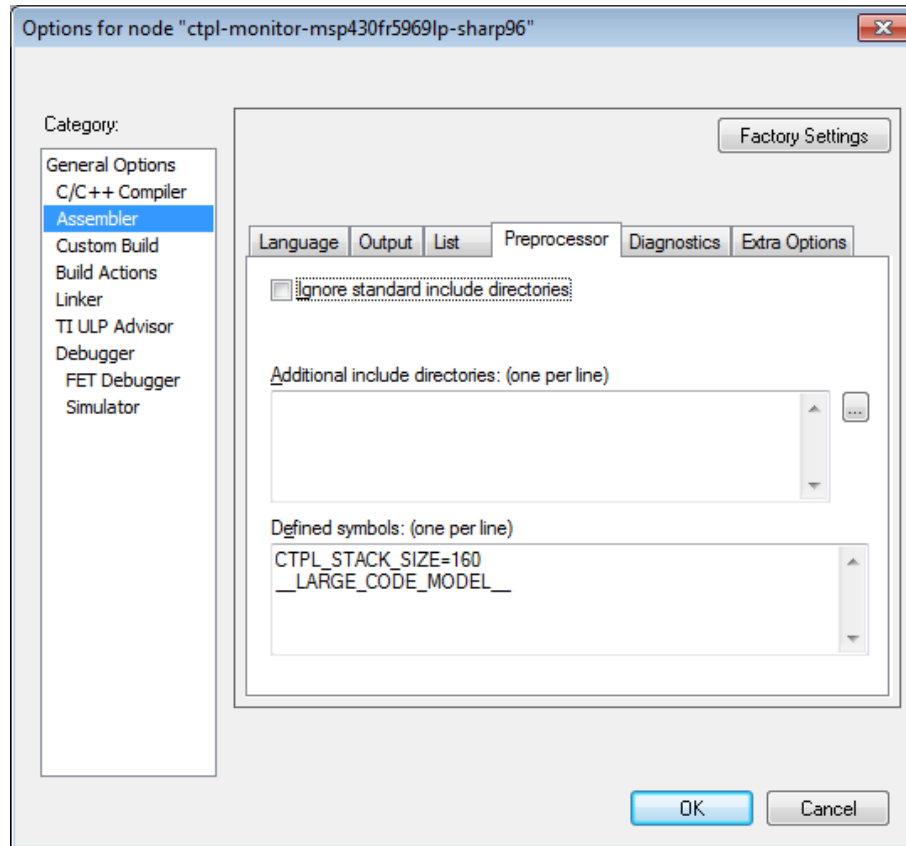


Figure 2.4: CTPL assembler options

4. Configure the project to use the device linker file (.xcl extension) in the project linker options.

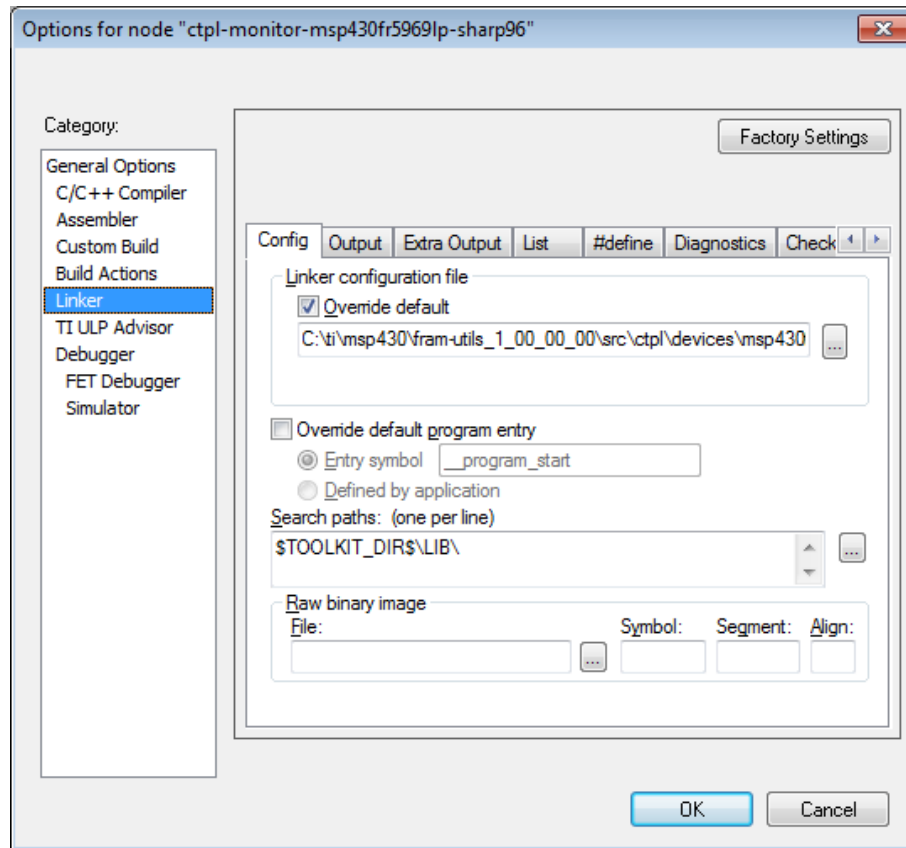


Figure 2.5: CTPL IAR linker file

## 2.3 API Reference

API Overview .....	15
Core API Set .....	15
Low Level .....	21
Peripherals .....	23
Benchmark .....	25

### 2.3.1 API Overview

The CTPL library is designed to provide a simplified [Core API set](#) for use by the application. Methods outside of this API set have been documented below but are not intended to be modified or directly interfaced with by the main application.

### 2.3.2 Core API Set

#### Defines

- [CTPL\\_DISABLE\\_RESTORE\\_ON\\_RESET](#)
- [CTPL\\_ENABLE\\_RESTORE\\_ON\\_RESET](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_1024\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_128\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_16\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_1\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_256\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_2\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_32\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_4\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_512\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_64\\_MS](#)
- [CTPL\\_SHUTDOWN\\_TIMEOUT\\_8\\_MS](#)

#### Functions

- void [ctpl\\_enterLpm35](#) (bool restoreOnReset)
- void [ctpl\\_enterLpm45](#) (bool restoreOnReset)
- void [ctpl\\_enterShutdown](#) (uint16\_t timeout)
- void [ctpl\\_init](#) (void)

#### 2.3.2.1 Detailed Description

The following is a reference of all CTPL API's available for the application to use. The application should only directly interface with the function defined in `ctpl/ctpl.h` and listed below.

## 2.3.2.2 Define Documentation

### 2.3.2.2.1 CTPL\_DISABLE\_RESTORE\_ON\_RESET

**Definition:**

```
#define CTPL_DISABLE_RESTORE_ON_RESET
```

**Description:**

Do not allow the CTPL utility to restore a saved state if the device is reset or powered on from a cold start.

### 2.3.2.2.2 CTPL\_ENABLE\_RESTORE\_ON\_RESET

**Definition:**

```
#define CTPL_ENABLE_RESTORE_ON_RESET
```

**Description:**

Allow the CTPL utility to restore a saved state if the device is reset or powered on from a cold start.

### 2.3.2.2.3 CTPL\_SHUTDOWN\_TIMEOUT\_1024\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_1024_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 1024 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

### 2.3.2.2.4 CTPL\_SHUTDOWN\_TIMEOUT\_128\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_128_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 128 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

### 2.3.2.2.5 CTPL\_SHUTDOWN\_TIMEOUT\_16\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_16_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 16 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.



**2.3.2.2.6 CTPL\_SHUTDOWN\_TIMEOUT\_1\_MS****Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_1_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 1 millisecond the watchdog timer will reset the device and cause a restore of the saved state.

**2.3.2.2.7 CTPL\_SHUTDOWN\_TIMEOUT\_256\_MS****Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_256_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 256 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

**2.3.2.2.8 CTPL\_SHUTDOWN\_TIMEOUT\_2\_MS****Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_2_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 2 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

**2.3.2.2.9 CTPL\_SHUTDOWN\_TIMEOUT\_32\_MS****Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_32_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 32 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

**2.3.2.2.10 CTPL\_SHUTDOWN\_TIMEOUT\_4\_MS****Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_4_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 4 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

### 2.3.2.2.11 CTPL\_SHUTDOWN\_TIMEOUT\_512\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_512_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 512 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

### 2.3.2.2.12 CTPL\_SHUTDOWN\_TIMEOUT\_64\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_64_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 64 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

### 2.3.2.2.13 CTPL\_SHUTDOWN\_TIMEOUT\_8\_MS

**Definition:**

```
#define CTPL_SHUTDOWN_TIMEOUT_8_MS
```

**Description:**

Timeout duration that can be passed to [ctpl\\_enterShutdown\(\)](#). If the device does not enter BOR after 8 milliseconds the watchdog timer will reset the device and cause a restore of the saved state.

## 2.3.2.3 Function Documentation

### 2.3.2.3.1 `ctpl_enterLpm35` Save state and enter into low power mode LPM3.5.

**Prototype:**

```
void  
ctpl_enterLpm35 (bool restoreOnReset)
```

**Description:**

LPM3.5 does not retain the settings of peripheral registers or RAM contents so these settings and states must be saved to non-volatile FRAM. This function will save the state of all the peripherals defined in the include device file, the context of the CPU and the active stack to non-volatile FRAM storage. After saving the state it is marked as valid so that it may be restored after wakeup and the function will enter into LPM3.5. When the device wakes up due to an interrupt or reset/power on event the [ctpl\\_init\(\)](#) function will check if the state is valid and if it should be restored. The `restoreOnReset` argument determines if state context is restored on a device reset or power on, passing true will always restore the saved state where as passing false will only restore state on a LPM3.5 wakeup from interrupt (returning to the start of main if the device was reset). The saved peripheral states, CPU states and stack are restored from the FRAM storage and the function returns back to the application from where it was called.

This function bypasses the need to check at device start up for a LPM3.5 wakeup and the application only needs to reinitialize peripherals that are not saved by the utility.

This API is functionally the same as [ctpl\\_enterLpm45\(\)](#). The actual low-power mode used (LPM3.5 or LPM4.5) is determined by the state of the RTC peripheral, LPM3.5 is used if the RTC is enabled and LPM4.5 is used if the RTC is disabled. For more information on low power modes refer to the device datasheet and user's guide.

**Parameters:**

**restoreOnReset** Allow the CTPL utility to restore a saved state if the device is reset or powered on from a cold start. Valid values are:

- **CTPL\_DISALLOW\_RESTORE\_ON\_RESET**
- **CTPL\_ALLOW\_RESTORE\_ON\_RESET**

**Returns:**

none

### 2.3.2.3.2 **ctpl\_enterLpm45** Save state and enter into low power mode LPM4.5.

**Prototype:**

```
void  
ctpl_enterLpm45 (bool restoreOnReset)
```

**Description:**

LPM4.5 does not retain the settings of peripheral registers or RAM contents so these settings and states must be saved to non-volatile FRAM. This function will save the state of all the peripherals defined in the include device file, the context of the CPU and the active stack to non-volatile FRAM storage. After saving the state it is marked as valid so that it may be restored after wakeup and the function will enter into LPM4.5. When the device wakes up due to an interrupt or reset/power on event the [ctpl\\_init\(\)](#) function will check if the state is valid and if it should be restored. The restoreOnReset argument determines if state context is restored on a device reset or power on, passing true will always restore the saved state where as passing false will only restore state on a LPM4.5 wakeup from interrupt (returning to the start of main if the device was reset). The saved peripheral states, CPU states and stack are restored from the FRAM storage and the function returns back to the application from where it was called. This function bypasses the need to check at device start up for a LPM4.5 wakeup and the application only needs to reinitialize peripherals that are not saved by the utility.

This API is functionally the same as [ctpl\\_enterLpm35\(\)](#). The actual low-power mode used (LPM3.5 or LPM4.5) is determined by the state of the RTC peripheral, LPM3.5 is used if the RTC is enabled and LPM4.5 is used if the RTC is disabled. For more information on low power modes refer to the device datasheet and user's guide.

**Parameters:**

**restoreOnReset** Allow the CTPL utility to restore a saved state if the device is reset or powered on from a cold start. Valid values are:

- **CTPL\_DISALLOW\_RESTORE\_ON\_RESET**
- **CTPL\_ALLOW\_RESTORE\_ON\_RESET**

**Returns:**

none

### 2.3.2.3.3 `ctpl_enterShutdown` Save the state when power is lost.

**Prototype:**

```
void  
ctpl_enterShutdown(uint16_t timeout)
```

**Description:**

Device shutdown does not retain the settings of peripheral registers or RAM contents so these settings and states must be saved to non-volatile FRAM. This function will save the state of all the peripherals defined in the include device file, the context of the CPU and the active stack to non-volatile FRAM storage. After saving the state it is marked as valid so that it may be restored after a reset or powering the device back on. All interrupt and wakeup sources are disabled and the device waits in active mode for the SVS to put the device into BOR. MCLK is configured to 4MHz and the SMCLK and WDT\_A dividers are set based on the timeout parameter. In this state the only source of a wakeup is a device reset, power up or a shutdown timeout. In all three wakeup scenarios the state is restored and the application resumes. The saved peripheral states, CPU states and stack are restored from the FRAM storage and the function returns back to the application from where it was called.

When configuring the shutdown timeout parameter the device supply voltage and ramp conditions should be considered to avoid scenarios where voltage ramps down too slowly. If the timeout duration is not long enough the timeout will trigger a restore before the device enters the BOR state. In this scenario the restored image is no longer valid and the next power on will cause a device reset to the beginning of the main application. To prevent this a timeout duration should be selected so that sufficient time is provided for the supply voltage to ramp down and the timeout only triggers in the scenario where voltage ramps back up to operational levels.

This API provides a method for application programmers to efficiently save the application state and shutdown the CPU when a power loss is detected and restore the applications state when the device regains power. The utility includes two examples demonstrating methods for monitoring the device voltage and detecting a power loss.

This API only saves and restores RTC\_B and RTC\_C registers that are not retained in LPMx.5 modes. In device shutdown the context of these other registers must be reinitialized if using these peripherals. See the device users guide for the complete list of RTC registers and details on which are retained.

**Parameters:**

***timeout*** Configurable timeout for a reset if device does not enter BOR. Valid values are:

- `CTPL_SHUTDOWN_TIMEOUT_1_MS`
- `CTPL_SHUTDOWN_TIMEOUT_2_MS`
- `CTPL_SHUTDOWN_TIMEOUT_4_MS`
- `CTPL_SHUTDOWN_TIMEOUT_8_MS`
- `CTPL_SHUTDOWN_TIMEOUT_16_MS`
- `CTPL_SHUTDOWN_TIMEOUT_32_MS`
- `CTPL_SHUTDOWN_TIMEOUT_64_MS`
- `CTPL_SHUTDOWN_TIMEOUT_128_MS`
- `CTPL_SHUTDOWN_TIMEOUT_256_MS`
- `CTPL_SHUTDOWN_TIMEOUT_512_MS`
- `CTPL_SHUTDOWN_TIMEOUT_1024_MS`

**Returns:**

none

#### 2.3.2.3.4 `ctpl_init` Initialize the CTPL utility.

**Prototype:**

```
void  
ctpl_init(void)
```

**Description:**

This function initializes the utility and must be called at the start of the `_system_pre_init` function for CCS or the `__low_level_init` function for IAR. By default these functions are defined in `ctpl_pre_init.c` but some applications might have their own version of the function. In this case the `ctpl_pre_init.c` file can be omitted and the function called at the start of the application's low level function.

**Returns:**

none

### 2.3.3 Low Level

**Defines**

- `CTPL_MODE_LPM35`
- `CTPL_MODE_LPM45`
- `CTPL_MODE_LPMX5_WAKEUP`
- `CTPL_MODE_NONE`
- `CTPL_MODE_RESTORE_RESET`
- `CTPL_MODE_SHUTDOWN`

**Functions**

- void `ctpl_saveCpuStackEnterLpm` (uint16\_t timeout)

**Variables**

- volatile uint16\_t `ctpl_mode`

#### 2.3.3.1 Detailed Description

The following is a reference of the CTPL low level functions. These functions are invoked by the [Core API Set](#) and should not be called from outside the utility.

#### 2.3.3.2 Define Documentation

##### 2.3.3.2.1 `CTPL_MODE_LPM35`

**Definition:**

```
#define CTPL_MODE_LPM35
```

**Description:**

Bits that define the LPM3.5 CTPL mode.

**2.3.3.2.2 CTPL\_MODE\_LPM45**

**Definition:**

```
#define CTPL_MODE_LPM45
```

**Description:**

Bits that define the LPM4.5 mode.

**2.3.3.2.3 CTPL\_MODE\_LPMX5\_WAKEUP**

**Definition:**

```
#define CTPL_MODE_LPMX5_WAKEUP
```

**Description:**

Bits that define the LPM3.5 and LPM4.5 wakeup flags.

**2.3.3.2.4 CTPL\_MODE\_NONE**

**Definition:**

```
#define CTPL_MODE_NONE
```

**Description:**

Bits that define no CTPL mode.

**2.3.3.2.5 CTPL\_MODE\_RESTORE\_RESET**

**Definition:**

```
#define CTPL_MODE_RESTORE_RESET
```

**Description:**

Bits that define the optional restoreOnReset flag.

**2.3.3.2.6 CTPL\_MODE\_SHUTDOWN**

**Definition:**

```
#define CTPL_MODE_SHUTDOWN
```

**Description:**

Bits that define the shutdown CTPL mode.

### 2.3.3.3 Function Documentation

#### 2.3.3.3.1 `ctpl_saveCpuStackEnterLpm` Low level assembly function used to save the state and enter LPM.

**Prototype:**

```
void  
ctpl_saveCpuStackEnterLpm(uint16_t timeout)
```

**Description:**

This assembly function saves the CPU state and stack into non-volatile FRAM before setting the state as valid and entering into the low-power mode defined by `ctpl_mode`. On device reset with a valid state `ctpl_init` will jump back to this function which restores the CPU state and stack from the FRAM copy. After restoring the state the function returns to the higher-level CTPL function that was invoked by the main application.

This function is only intended to be called from within the library code, the user does not need to invoke this function manually.

**Parameters:**

***timeout*** Configurable timeout for a reset if device does not enter BOR. Valid values are:

- `CTPL_POWERLOSS_TIMEOUT_1_MS`
- `CTPL_POWERLOSS_TIMEOUT_2_MS`
- `CTPL_POWERLOSS_TIMEOUT_4_MS`
- `CTPL_POWERLOSS_TIMEOUT_8_MS`
- `CTPL_POWERLOSS_TIMEOUT_16_MS`
- `CTPL_POWERLOSS_TIMEOUT_32_MS`
- `CTPL_POWERLOSS_TIMEOUT_64_MS`
- `CTPL_POWERLOSS_TIMEOUT_128_MS`
- `CTPL_POWERLOSS_TIMEOUT_256_MS`
- `CTPL_POWERLOSS_TIMEOUT_512_MS`
- `CTPL_POWERLOSS_TIMEOUT_1024_MS`

**Returns:**

none.

### 2.3.3.4 Variable Documentation

#### 2.3.3.4.1 `ctpl_mode`

**Definition:**

```
volatile uint16_t ctpl_mode
```

**Description:**

CTPL mode variable set by the utility and used to determine the mode before low-power mode or shutdown as well as the flags set for the wakeup source.

## 2.3.4 Peripherals

### Data Structures

- `ctpl_peripheral`

## Variables

- const `ctpl_peripheral * ctpl_peripherals[]`
- const `uint16_t ctpl_peripheralsLen`

### 2.3.4.1 Detailed Description

The following is a reference of the CTPL peripheral functions. These functions are invoked by the [Core API Set](#) and should not be called from outside the utility.

### 2.3.4.2 Data Structure Documentation

#### 2.3.4.2.1 `ctpl_peripheral`

##### Definition:

```
typedef struct
{
    uint16_t baseAddress;
    uint16_t *storage;
    ctpl_tFunction save;
    ctpl_tFunction restore;
    ctpl_tFunction epilogue;
}
ctpl_peripheral
```

##### Members:

***baseAddress*** Peripheral base address.

***storage*** Peripheral non-volatile storage.

***save*** Function to save peripheral context.

***restore*** Function to restore peripheral context.

***epilogue*** Optional function to run after clearing the LOCKLPM5 bit. If this function pointer is null the function will not be called.

##### Description:

Structure defining how to save and restore a peripherals context. These structures are provided for each device in the included device-specific `ctpl_*.c` file required when using the utility.

### 2.3.4.3 Variable Documentation

#### 2.3.4.3.1 `ctpl_peripherals`

##### Definition:

```
const ctpl_peripheral *ctpl_peripherals[]
```

##### Description:

The device specific array of peripherals to save and restore. This symbol is defined in the device-specific `ctpl_*.c` file included with the library.



### 2.3.4.3.2 ctpl\_peripheralsLen

**Definition:**

```
const uint16_t ctpl_peripheralsLen
```

**Description:**

Abstracted symbol for the length of the ctpl\_peripherals array. This symbol is defined in the device-specific ctpl\_\*.c file required when using the library.

## 2.3.5 Benchmark

**Defines**

- [CTPL\\_BENCHMARK\\_DIR](#)
- [CTPL\\_BENCHMARK\\_OUT](#)
- [CTPL\\_BENCHMARK\\_PIN](#)

### 2.3.5.1 Detailed Description

The following is a reference of the CTPL benchmark function. These defines are used by the [Core API Set](#) and should not be referenced from outside the utility.

### 2.3.5.2 Define Documentation

#### 2.3.5.2.1 CTPL\_BENCHMARK\_DIR

**Definition:**

```
#define CTPL_BENCHMARK_DIR
```

**Description:**

Benchmark port direction register used when CTPL\_BENCHMARK is defined in the compiler settings (-DCTPL\_BENCHMARK).

#### 2.3.5.2.2 CTPL\_BENCHMARK\_OUT

**Definition:**

```
#define CTPL_BENCHMARK_OUT
```

**Description:**

Benchmark port output register used when CTPL\_BENCHMARK is defined in the compiler settings (-DCTPL\_BENCHMARK).

#### 2.3.5.2.3 CTPL\_BENCHMARK\_PIN

**Definition:**

```
#define CTPL_BENCHMARK_PIN
```

**Description:**

Benchmark pin used when CTPL\_BENCHMARK is defined in the compiler settings (-DCTPL\_BENCHMARK).

## 2.4 Examples

Examples Overview .....27  
 LPM4.5 With GPIO Wakeup .....28  
 LPM3.5 With RTC Wakeup .....29  
 COMP\_E Powerloss Monitor .....30  
 ADC12\_B Powerloss Monitor .....31

### 2.4.1 Examples Overview

These examples demonstrate how to use the CTPL utility in several application use cases. The examples are implemented for all FRAM LaunchPad Development Kits and Experimenter Boards. See table below for supported hardware and examples.

Hardware	Examples			
	LPM4.5 GPIO	LPM3.5 RTC	COMP_E Powerloss	ADC12_B Powerloss
msp-exp430fr5739	✓	✗	✗	✗
msp-exp430fr5969	✓	✓	✓	✓
msp-exp430fr6989	✓	✓	✓	✓

Table 2.1: Hardware support for CTPL examples

Using CCS and Resource Explorer it's easy to import and run the examples. Navigate to the CCS "View" menu and select "Resource Explorer (Examples)". Under the MSPWare package libraries select the FRAM-Utilities node and then CTPL node to view examples, user guides and release notes.

## 2.4.2 LPM4.5 With GPIO Wakeup

This example is an adaptation of the C code example `mcp430fr59xx_lpm4-5_01` and demonstrates how to enter LPM4.5 and wakeup from a GPIO interrupt. The example will turn on P4.6 and enter into LPM4.5. When P1.1 (S2 on MSP-EXP430FR5969) transitions from high to low the example will turn off P4.6 to indicate the device is no longer in LPM4.5 and blink P1.0 forever.

By using the compute through power loss (CTPL) library the original example code is greatly simplified. The peripherals are initialized once at the start of the application and the library will save the peripheral and application state in FRAM before entering LPM. Upon wakeup from LPM the peripheral and application state is restored and the code continues execution from the next line of code.

```
// ACLK = VLOCLK, MCLK = SMCLK = DCO = ~1MHz
//
//           MSP-EXP430FR5969
//           -----
//           /|\|           |
//           | |           P1.0|----> LED2
//           --|RST       P4.6|----> LED1
//           |           |
//           |           P1.1|<--- S2 push-button
//           |           |
```

### 2.4.3 LPM3.5 With RTC Wakeup

This example is an adaptation of the C code example `msp430fr59xx_lpm3-5_02` and demonstrates how to use `RTC_B` as an interval wakeup in LPM3.5. The example will toggle P4.6 after initialization to indicate a device start up and then enter LPM3.5 with interrupts enabled. The RTC interrupt will wake the device up every two seconds and toggle P1.0.

By using the compute through power loss (CTPL) library the original example code is greatly simplified. The peripherals are initialized once at the start of the application and the library will save the peripheral and application state in FRAM before entering LPM. Upon wakeup from LPM the peripheral and application state is restored and the code continues execution from the next line of code.

```
// ACLK = 32.768kHz, MCLK = SMCLK = DCO = ~1MHz
//
//      MSP-EXP430FR5969
//      -----
//      /|\|          XIN|-
//      | |          | 32kHz
//      --|RST      XOUT|-
//      |          |
//      |          P1.0|--> LED2
//      |          P4.6|--> LED1
//      |          |
```

## 2.4.4 COMP\_E Powerloss Monitor

This example demonstrates how to use the COMP\_E peripheral and an external voltage divider to actively monitor supply voltage and detect when power is lost. The comparator is configured with a 1.5V reference and an external voltage divider provides  $V_{cc}/2$  to the input pin (P1.5/C5). When  $V_{cc}/2$  drops below the 1.5V reference (meaning  $V_{cc}$  is below 3.0V) the comparator interrupt service routine will disable the comparator monitor and invoke the `ctpl_enterShutdown` API. This API will save the application and peripheral state and waits for the device to enter BOR with a 64ms timeout. The device will restore application and peripheral state when power is restored and continue execution from the next line of code.

The main application will blink LED2 with incremental counts, resetting after four blinks. The power supply can be removed (by disconnecting the USB cable or unplugging the jumpers connecting the on-board emulator to the device) after a specific count of blink and then reapplied to verify that context was saved.

```
// ACLK = VLOCLK, MCLK = SMCLK = DCO = ~1MHz
//
//           MSP-EXP430FR5969
//           -----
//           /|\|           P1.7|----> Vcc
//           | |           (C5)P1.5|----> Vcc/2 (350k/350k voltage divider)
//           --|RST       P1.4|----> GND
//           |           |
//           |           P1.0|----> LED2
//           |           |
```

## 2.4.5 ADC12\_B Powerloss Monitor

This example demonstrates how to use the ADC12\_B battery monitor and window comparator to actively monitor supply voltage and detect when power is lost. The ADC12\_B peripheral is configured with a 2.0V reference voltage and the internal battery monitor channel provides  $V_{cc}/2$ . The ADC12\_B low side window comparator is configured to trigger the interrupt when  $V_{cc}$  reaches ADC\_MONITOR\_THRESHOLD, 3.0V by default. The high side window comparator is set to ADC\_MONITOR\_THRESHOLD + 0.1V to ensure the device has reached a stable voltage before enabling the monitor. When the high side interrupt is triggered it is disabled and the low side interrupt is enabled to begin actively monitoring  $V_{cc}$ . When power loss is detected the device will invoke the `ctpl_enterShutdown` API which saves the application and peripheral state and waits for the device to enter BOR with a 64ms timeout. The device will restore application and peripheral state when power is restored and continue execution from the next line of code.

The main application will blink LED2 with incremental counts, resetting after four blinks. The power supply can be removed (by disconnecting the USB cable or unplugging the jumpers connecting the on-board emulator to the device) after a specific count of blink and then reapplied to verify that context was saved.

```
// ACLK = VLOCLK, MCLK = SMCLK = DCO = ~1MHz
//
//      MSP-EXP430FR5969
//      -----
//      /|\|          |
//      | |          |
//      --|RST      P1.0|----> LED2
//      |          |
//      |          |
```

## 2.5 Benchmarking

Overview ..... 32  
 Configuration ..... 34

### 2.5.1 Overview

The CTPL utility can be benchmarked by defining CTPL\_BENCHMARK in the compiler and assembler predefined symbols. When this symbol is defined the code will toggle a single pin to indicate the CTPL function has started. Once the state has been saved the software will toggle the benchmark pin to indicate the end of the CTPL function. The `ctpl_enterShutdown()` function will continue to toggle the benchmark pin inside the software loop while waiting for the device to enter a BOR. The repeated pin toggle provides a measurement of how long the CPU can run before complete power is lost and the device shuts down to help select the right timeout parameter.

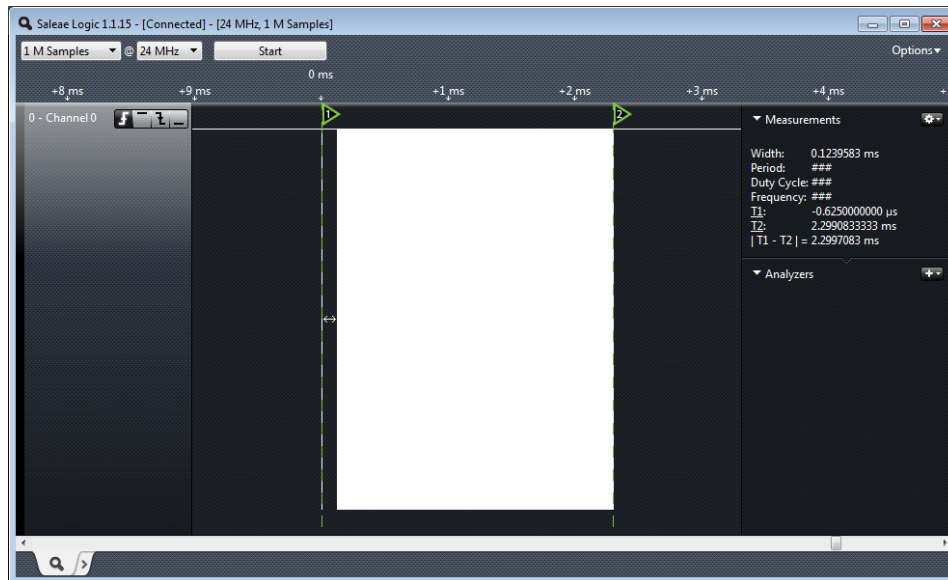


Figure 2.6: Benchmark of the `ctpl_enterShutdown()` function when power is lost (8MHz CPU clock)

The above screen capture shows the `ctpl_enterShutdown()` API when power is lost on a MSP430FR5969 device running at 8MHz with all available peripherals saved (a total of eleven, see [peripheral usage](#) section for the complete peripheral list). In this example the "Width" measurement is the total time the API needs to save the state of the peripherals, stack and CPU. The " $|T1 - T2|$ " measurement indicates the life of the CPU before complete power is lost. The second measurement will be dependant on both the hardware design and the software configuration of the device (active peripherals when entering API). In scenarios where power is lost it's best practice to shut down any active peripherals before calling the API to conserve the remaining energy.



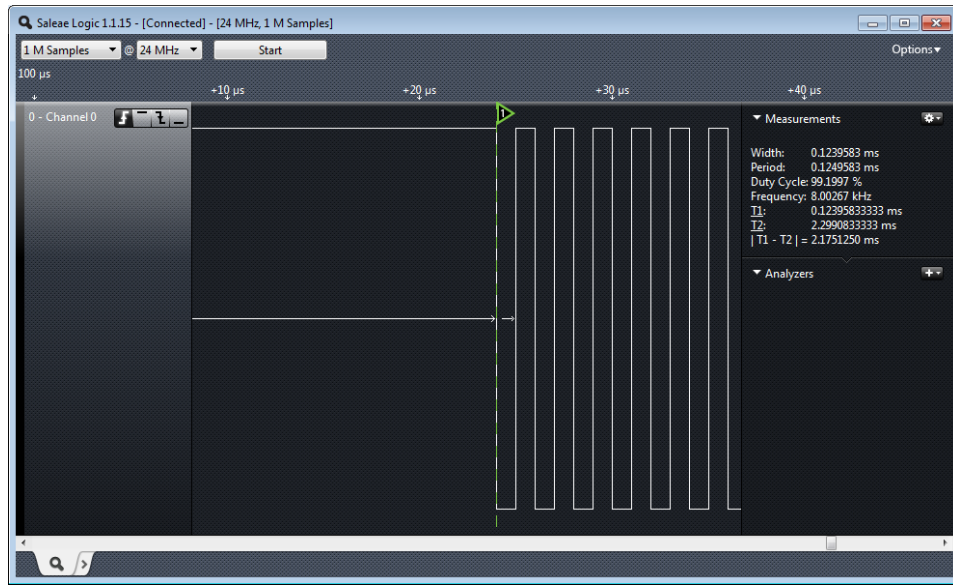


Figure 2.7: Close up view of benchmark pin toggle

## 2.5.2 Configuration

The pin used for the benchmark is defined in `ctpl_benchmark.h`. By default P4.6 is used, LED1 on the MSP430-EXP430FR5969 LaunchPad. This pin can be change to any available GPIO by editing this file and changing the pin and port registers used.

See the [Benchmark API reference](#) for more on configuration.



---

# IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

## Products

Amplifiers	<a href="http://amplifier.ti.com">amplifier.ti.com</a>
Data Converters	<a href="http://dataconverter.ti.com">dataconverter.ti.com</a>
DLP® Products	<a href="http://www.dlp.com">www.dlp.com</a>
DSP	<a href="http://dsp.ti.com">dsp.ti.com</a>
Clocks and Timers	<a href="http://www.ti.com/clocks">www.ti.com/clocks</a>
Interface	<a href="http://interface.ti.com">interface.ti.com</a>
Logic	<a href="http://logic.ti.com">logic.ti.com</a>
Power Mgmt	<a href="http://power.ti.com">power.ti.com</a>
Microcontrollers	<a href="http://microcontroller.ti.com">microcontroller.ti.com</a>
RFID	<a href="http://www.ti-rfid.com">www.ti-rfid.com</a>
RF/IF and ZigBee® Solutions	<a href="http://www.ti.com/lprf">www.ti.com/lprf</a>

## Applications

Audio	<a href="http://www.ti.com/audio">www.ti.com/audio</a>
Automotive	<a href="http://www.ti.com/automotive">www.ti.com/automotive</a>
Broadband	<a href="http://www.ti.com/broadband">www.ti.com/broadband</a>
Digital Control	<a href="http://www.ti.com/digitalcontrol">www.ti.com/digitalcontrol</a>
Medical	<a href="http://www.ti.com/medical">www.ti.com/medical</a>
Military	<a href="http://www.ti.com/military">www.ti.com/military</a>
Optical Networking	<a href="http://www.ti.com/opticalnetwork">www.ti.com/opticalnetwork</a>
Security	<a href="http://www.ti.com/security">www.ti.com/security</a>
Telephony	<a href="http://www.ti.com/telephony">www.ti.com/telephony</a>
Video & Imaging	<a href="http://www.ti.com/video">www.ti.com/video</a>
Wireless	<a href="http://www.ti.com/wireless">www.ti.com/wireless</a>

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265  
Copyright © , Texas Instruments Incorporated