AN1398: EFR32 Coulomb Counting

This application note describes the configuration, calibration, and operation of the Coulomb Counter on the EFR32 Series 2 devices. The integrated Coulomb Counter can losslessly measure the charge drawn from the DC-DC of the selected EFR32 device. A software example demonstrating the coulomb counting feature is included in the Gecko SDK

KEY POINTS

- Ability to losslessly measure charge drawn from EFR32 DC-DC.
- Internally integrated calibration registers for accurate calibration.
- Example C-code provided in Gecko SDK

1. Device Compatibility

This application note supports devices in the EFR32 Wireless SoC product family which contain DC-DC and coulomb counting hardware.

Supported devices include:

- EFR32BG27
- EFR32MG27

2. Coulomb Counter

The DC-DC on selected EFR32 devices includes two 32-bit counters that are used to measure the charge delivered by the DC-DC. The Coulomb Counter circuit counts the number of charge pulses delivered by the DC-DC. Software can determine the charge-per-pulse (CPP) using on-chip resources, and keep track of the total charge delivered. Separate counters are used to measure pulses delivered in EM0/1 and EM2/3. Each counter provides threshold and overflow interrupts to minimize system software overhead. Measured on-chip loads are available for periodic recalibration.

2.1 Startup

The Coulomb Counter hardware is disabled following any power-on-reset, but will remain active through any other reset source.

Software should begin by configuring the counter thresholds EM0CNT and EM2CNT in DCDC_CCTHR while the counter is still disabled. These 16-bit thresholds are compared against the 16 MSBs of the respective EM0/1 and EM2/3 counters, and can be used to establish a service interval for the Coulomb Counter hardware. When the 16 MSBs of a counter exceed the programmed threshold, the EM0CMPIF or EM2CMPIF interrupt flag in DCDC_CCIF will be set and an interrupt may optionally be generated. As a guideline, using the default DC-DC settings, the DC-DC may produce up to 60,000-70,000 pulses per second for every mA of load current. Therefore, using a threshold value of 0x00001 in the DCDC_CCTHR.EM0CNT register will cause an interrupt after roughly 1 mC of charge has been delivered in EM0 and EM1.

After the thresholds have been programmed to the desired service intervals, the counters should be enabled by writing 1 to DCDC_CCCMD.START. The thresholds cannot be changed while the Coulomb Counter is running, and it must be halted using the DCDC_CCMD.STOP command if it is required to change thresholds.

Once the counter is running, software can perform an initial calibration to more precisely measure the charge-per-pulse delivered by the DC-DC in EM0/1 and EM2/3 (CPP_{EM01} and CPP_{EM23}) See 2.2 Calibration for more detail. The CPP values may be stored in nonvolatile memory to withstand any device resets and keep track of the energy usage. Periodic re-calibration may also be performed with the counter running.

2.2 Calibration

The coulomb counter should be calibrated to determine the charge per pulse, both for EM0/1 and for EM2/3 settings of the DC-DC. The charge per pulse is measured using known on-chip calibration loads, a PRS channel, and the CMU RC oscillator calibration circuitry. A high-frequency oscillator source of known frequency (f_{HF}) is used to time pulses from the DC-DC and establish the charge per pulse. To cancel out any system current, two loads of different magnitude are used in the calibration procedure as follows:

1. Enable a larger calibration load (i_{load_lg}) and measure the number of high-frequency clocks (N_{lg}) to capture a known number of pulses (N_{cal})
2. Enable a smaller calibration load (i_{load_sm}) and measure the number of high-frequency clocks (N_{sm}) to capture a known number of pulses (N_{cal})
3. Calculate charge-per-pulse (CPP) using Figure 2.1 Charge-Per-Pulse (CPP) Calculation on page 4

$$CPP = [(N_{lg} * N_{sm}) / (f_{HF} * N_{cal})] * [(i_{load\_lg} - i_{load\_sm}) / (N_{sm} - N_{lg})]$$

Figure 2.1. Charge-Per-Pulse (CPP) Calculation

CPP should be calculated separately for the DC-DC EM0/1 and EM2/3 settings because these use different inductor peak current and timing parameters. Although the calibration must be performed in EM0, the DC-DC EM2/3 settings can be applied temporarily by setting the CCCALEM2 bit in DCDC_CCCALCTRL. CCCALEM2 must be cleared by software once the EM2/3 calibration is complete. Note that while the EM2/3 peak current settings are temporarily applied to the DC-DC converter, the maximum output current of the DC-DC will be extremely limited, and cannot support high current events (such as a radio transmission). For this reason, it is recommended to schedule EM2/3 calibration before EM2/3 entry, after all radio RX/TX communications.

The highest accuracy for the CPP measurement will be achieved by using the most accurate reference clock source available (generally HFXO), and taking measurements long enough to ensure timing precision. Both measurements should be taken while the system load is stable. If the system current is significantly different between the two measurements, it will impact the calibration accuracy. It is recommended to perform calibrations when the radio is not acive, and remain in a software polling loop or use the WFI instruction to wait for a CMU calibration complete interrupt.

The coulomb counter provides a software mechanism to halt calibration and identify unanticipated current events. The CCCALHALT bit in DCDC_CCCALCTRL may be set by ISR code or by RAIL software to halt a calibration. Setting CCCALHALT will remove any EM2/3 override (i.e., the DC-DC converter will revert to its EM0/1 peak current setting) and disable the calibration load current. The CMU calibration circuit will not stop, but software should read the CCCALHALT bit when calibration is complete (i.e., the CMU_IF.CALRDY IRQ flag is set) to determine if the measurement has been compromised in this way.

The number of pulses to capture (N_{cal}) and total load current both impact the required calibration time. If the HFXO is used as the up counter timing reference, N_{cal} = 2 for EM0/1 settings and N_{cal} = 8 for EM2/3 settings will generally provide sufficient timing precision for most loads. If the total system current during calibration is known to be higher, N_{cal} can be increased. However, if N_{cal} is set too high under light loads, the 20-bit calibration up counter may saturate during the measurement.

2.2.1 Calibration Load

The EFR32 includes a programmable calibration load current with eight settings, ranging from 0.25 to 8 mA. The calibration load magnitude is controlled by the CCLVL field and enabled using the CCLOADEN bit in DCDC_CCCALCTRL. While calibrating the coloumb counter, two different loads can be chosen to provide a known difference in the current and arrive at a more precise measurement. Each calibration load setting is measured during production test and the values are written to flash in the DEVINFO space as a 16-bit value with each LSB representing 200 nA. The stored value can be converted into uA by dividing by 5, or into mA by dividing by 5000. The load settings and corresponding DEVINFO locations are shown in Table 2.1 Calibration Load Current Settings on page 5.

Table 2.1. Calibration Load Current Settings

CCLVL	Nominal Load (mA)	DEVINFO Measurement Location	Nominal CCLOADx value (decimal)
LOAD0	0.25	CCLOAD10.CCLOAD0	1250
LOAD1	0.5	CCLOAD10.CCLOAD1	2500
LOAD2	1.0	CCLOAD32.CCLOAD2	5000
LOAD3	1.5	CCLOAD32.CCLOAD3	7500
LOAD4	2.0	CCLOAD54.CCLOAD4	10000
LOAD5	4.0	CCLOAD54.CCLOAD5	20000
LOAD6	6.0	CCLOAD76.CCLOAD6	30000
LOAD7	8.0	CCLOAD76.CCLOAD7	40000

Note: Calibration loads are applied at the DC-DC output, and the total load current during calibration will include all device current.

2.2.2 Pulse Timing

EFR32 devices include 20-bit up/down counters in the CMU intended for precise measurement of RC oscillators, which are described in the RC Oscillator Calibration section of the device reference manual. This circuit can be used to measure the time it takes for the DC-DC to produce a specific number of pulses. To accomplish this, the DCDC MONO70NSANA signal can be routed via PRS to the DOWN counter, and a precise clock source (HFXO for example) should be selected as the UP counter clock source. The DOWN counter is configured with a specified number of pulses to count using CALCTRL.CALTOP.

The steps for measuring pulse timing are as follows:

1. Configure an asynchronous PRS channel to route the DCDC MONO70NSANA producer to the CMU CALDN consumer
2. Configure the CMU calibration counter:
    a. CMU_CALCTRL.DOWNSEL = PRS
    b. CMU_CALCTRL.UPSEL = HFXO (recommended)
    c. CMU_CALCTRL.CONT = 0
    d. CMU_CALCTRL.CALTOP = number of DC-DC pulses to count (N_{cal} in the CPP equation)
3. Enable the desired calibration load:
    a. DCDC_CCCALCTRL.CCLVL = desired calibration load (i_{load_lg} or i_{load_sm} in the CPP equation)
    b. DCDC_CCCALCTRL.CALEN = 1
4. Wait for at least one DC-DC pulse to settle DC-DC:
    a. Clear DCDC_IF.REGULATIONIF
    b. Wait for DCDC_IF.REGULATIONIF == 1
5. Clear any pending calibration flag and start the measurement:
    a. CMU_IF_CLR.CALRDY = 1
    b. CMU_CALCMD.CALSTART = 1
6. Wait for the calibration to complete by polling CMU_IF.CALRDY, or using WFI with an ISR
7. Check for errors:
    a. If DCDC_CCCALCTRL.CCCALHALT == 1, the calibration was halted, and should be re-tried
    b. If CMU_CALCNT == 0xFFFFF, the specified number of DC-DC pulses was not seen before counter saturation (CMU_CALCTRL.CALTOP is too large)
8. If there are no errors, store the CMU_CALCNT value. This represents the number of high-frequency clocks required (N_{lg} or N_{sm} in the CPP equation)
9. Disable the calibration load by clearing DCDC_CCCALCTRL.CALEN = 0

2.3 Recalibration

Changes in external and internal conditions of the device can affect the accuracy of the previously determined charge-per-pulse (CPP). For example, recalibration may be required to maintain an accurate CPP after the following events:

- Significant change in input supply voltage
- Significant change in temperature
- Change of output voltage
- Change of ouput peak current
- Change in DC-DC operating mode
- Change in energy mode (e.g. EM0 -> EM2)

The Coulomb Counter Driver will be able to report when events like change in DC-DC operating mode is detected by using the API

```
sl_coulomb_counter_output_mask_t sl_coulomb_counter_outputs_need_calibration (void)
```

The application can then perform recalibration to refine the Coulomb Counter result.

The IADC or ACMP also may be used to periodically monitor the supply voltage for changes, and the EMU temperature sensor is available to monitor changes in temperature.

2.4 Servicing the Counters

The Coulomb Counter can be serviced infrequently, using either the built-in hardware charge thresholds and interrupt or a defined time interval from an RTC or timer. When it is time to service the counter, the present values representing the number of pulses should be read from DCDC_EM0CNT and DCDC_EM2CNT. Total charge delivered during this interval can be calculated as CPP_{EM01} * DCDC_EM0CNT + CPP_{EM23} * DCDC_EM2CNT, and stored in nonvolatile memory. After the charge has been calculated, it is recommended to clear the counters using the DCDC_CCCMD.CLR command.

3. Configuration using the Coulomb Counter Driver

With Simplicity Studio V5 and latest GSDK support, the user can configure the Coulomb Counter on EFR32 devices using the Coulomb Counter Driver provided in the GSDK Suite. This driver provides APIs that allow user to easily configure the Coulomb Counter without explicitly configuring the registers and worrying about specific details in the configuration. It is strongly recommended that users utilize the driver to correctly configure the Coulomb Counter for optimized operation.

The Coulomb Counter APIs documentation can be found in Coulomb Counter API. The documentation also provides a typical Coulomb Counter configuration routine.

The Coulomb Counter source code can be found in Coulomb Counter Source.

An out of the box software example demonstrating the Coulomb Counter on EFR32xG27 device is also provided within the GSDK. Section 4. Software Example provides more details regarding this example.

4. Software Example

Coulomb Counting with EFR32xG27

The GSDK Suite provides a software example that demonstrates the coulomb counting feature on the EFR32MG27 radio board BRD4194A. This example uses the integrated Coulomb Counter to monitor charge and current consumption on the DC-DC. It will store the information in the NVM3 region, and it will output results to VCOM console upon request from the CLI interface.

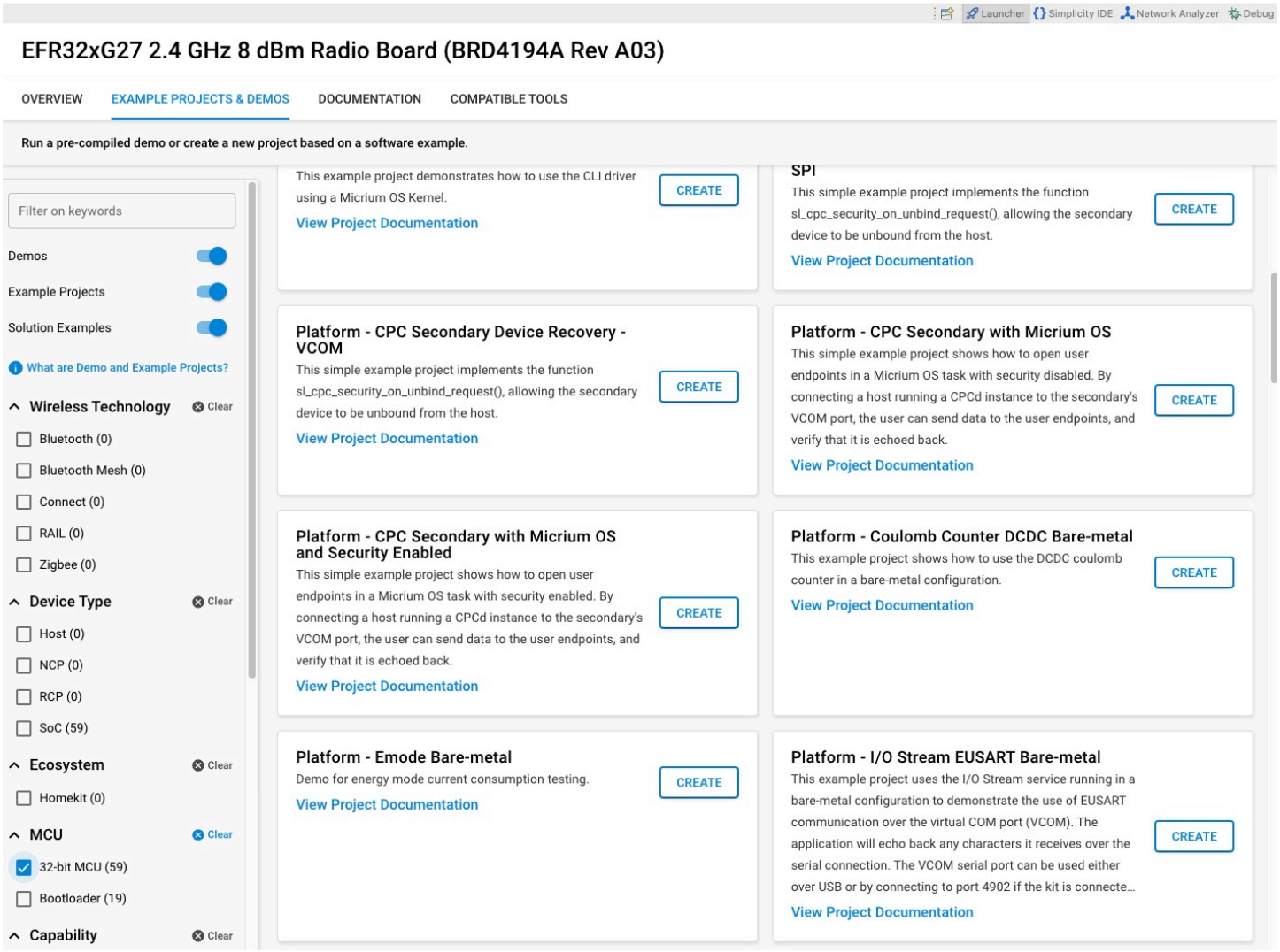
At the time of this application note authoring, the latest Simplicity Studio and GSDK version is:

- Simplicity Studio v5.6.3.1.
- Gecko SDK Suite v4.2.2.

This example is also available in Gecko SDK Suite v4.2.0+ and is expected to be available in future GSDK releases.

To run the example, you will also need a Wireless Starter Kit or Wireless Pro Kit + BRD4194A radio board.

The example name is *Platform - Coulomb Counter DCDC Bare-metal*, and can be found under the 32-bit MCU filter .



To run the example:

- Create and import the project into your workspace.
- Build the project to generate the hex image.
- Flash the hex image to BRD4194A, or alternatively you can enter debug mode to flash the device as well.
- Open serial terminal using desired software, or alternatively you can use the console built into Simplicity Studio.
- Reset the device by pressing the reset button, and you can enter CLI commands via the serial terminal interface. The readme.md file shows all the valid commands:
  - **coulomb_get**: Return coulombs consumed since last boot.
  - **coulomb_calibrate**: Calibrate coulomb counter.
  - **coulomb_get_total**: Return coulombs consumed over device's lifetime.
  - **coulomb_update_total**: Read coulomb counters and update total value.
  - **coulomb_reset_total**: Reset total counter to zero.

5. Related Documents

- AN0002.2: EFM32 and EFR32 Wireless Gecko Series 2 Hardware Design Considerations
- AN0948.2: EFM32 and EFR32 Series 2 DC-to-DC Converter
- EFR32xG27 Reference Manual
- Coulomb Counter API for GSDK v4.2.2

6. Revision History

Revision 0.1

May 2023

- Initial Revision.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com