Automatic LED current ripple tuning with CCM Buck converters

XMC1000

About this document

Scope and purpose

This document presents a method and algorithm that automatically adjusts the LED current ripple in a Continuous Conduction Mode (CCM) Buck LED driver solution with an XMC1000 microcontroller according to changes in supply voltage and LED forward voltage, to maintain average LED current.

Intended audience

This document is intended for engineers who wish to implement LED lighting applications with the XMC1000 microcontroller.

Applicable Products

- XMC120x
- XMC130x
- XMC140x
- DAVE™

References

Infineon: DAVE™, http://www.infineon.com/DAVE

Infineon: XMC™ Family, http://www.infineon.com/XMC

Infineon: AN_201609_PL30_028 – Driving LEDs with XMC1000, http://www.infineon.com/XMC1000

Infineon: XMC™ LED Current Control Explorer Kit, http://www.infineon.com/xmc_led_cc_exp

Table of contents

1 Introduction

Continuous Conduction Mode (CCM) Buck is a popular method of secondary-side LED driving because of its efficiency. A feature of the CCM buck method is the LED current ripple, which is an effect of the controller turning the MOSFET on and off in order to get a stable average current at a required value (Figure 1).

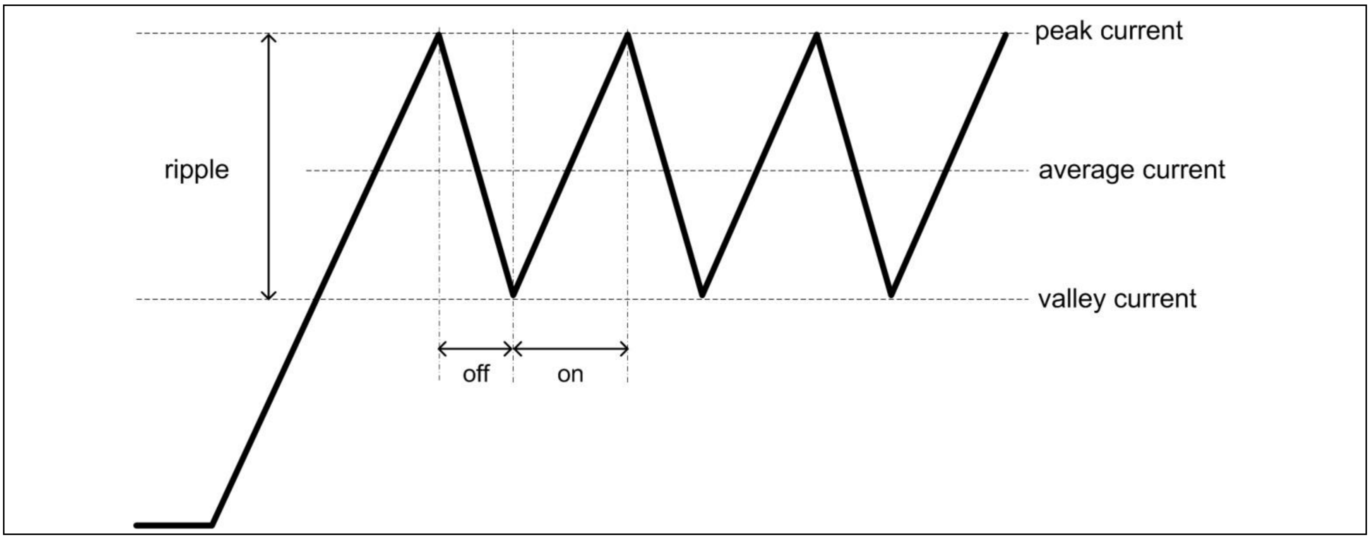

Figure 1 LED current ripple

One implementation method is to use two comparators to detect the peak and valley current respectively. The low-cost solution with XMC1000 microcontrollers uses an inverted buck configuration and implements a 'single-comparator-based peak-current detection with fixed off time' control of the LED current (Figure 2) per LED string. This means that the comparator detects the peak current level, then MOSFET is turned off for a pre-determined period of time. This is defined as 'off time', and it determines the ripple size and therefore the average current value.

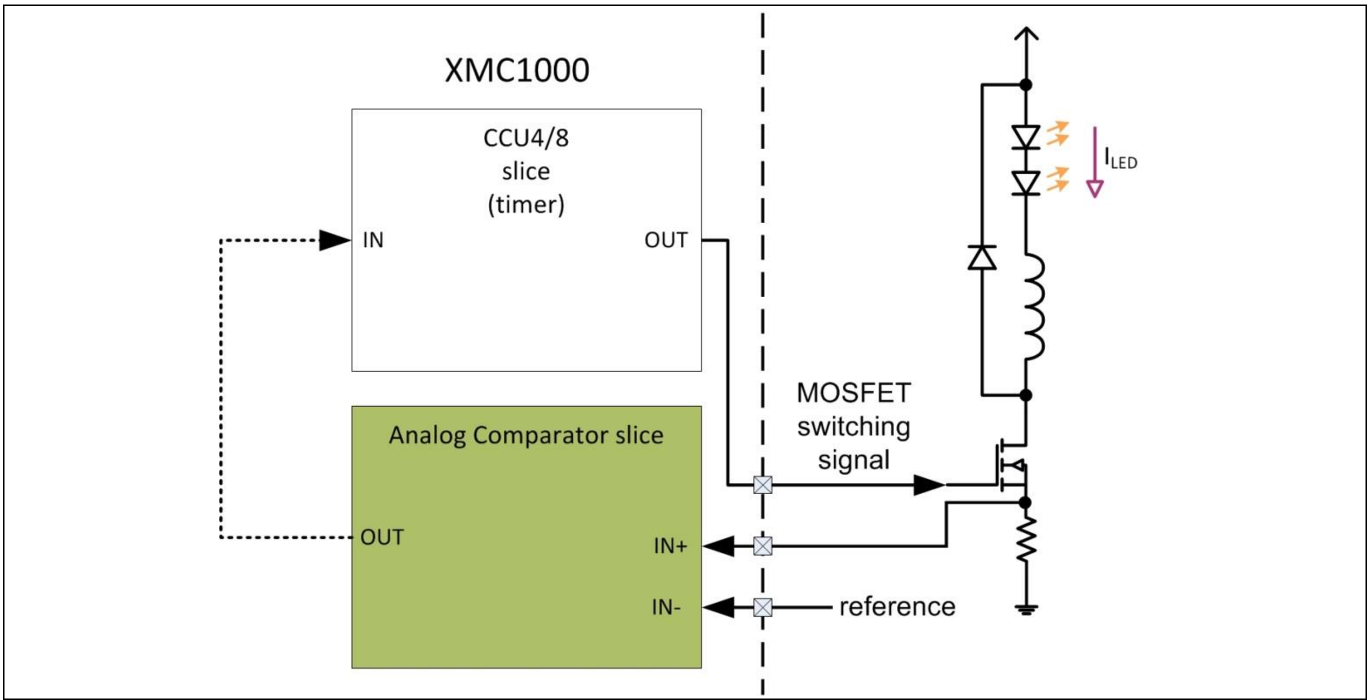

Figure 2 Single-comparator-based peak-current control circuitry

With a couple of measurements and a software algorithm, a method can be devised to re-calculate the required 'off time' in order to maintain the average LED current regardless of changes to the LED supply voltage and LED forward voltage. We explain the details of this method in this document.

Note: More information on the 'peak-current detection with fixed off time' control can be found in the application note AN_201609_PL30_028.

2 Method

In addition to the microcontroller resources required to implement the CCM Buck LED driver (as documented in AN_201609_PL30_028), two Analog-to-Digital Converter (ADC) channels are required; one to measure the input voltage (V_{in}) and the other to measure the inductance voltage (V_L) on the channel (Figure 3). Any additional LED channels will require an additional ADC channel to measure the channel's V_L. The XMC130x microcontrollers can drive up to 3 LED channels, and the XMC140x variants can drive up to 4 LED channels.

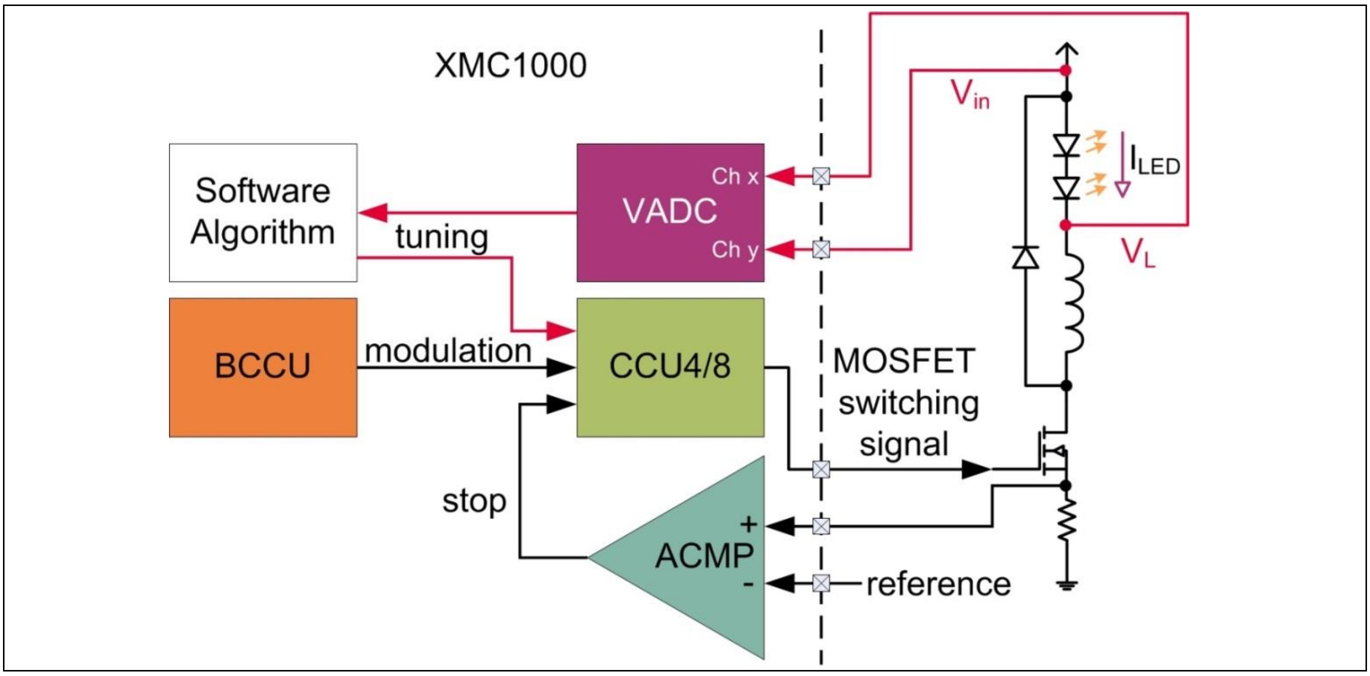

Figure 3 XMC1000 resources required

Figure 4 shows the LED current ripple and introduces the parameters involved in the ripple tuning process.

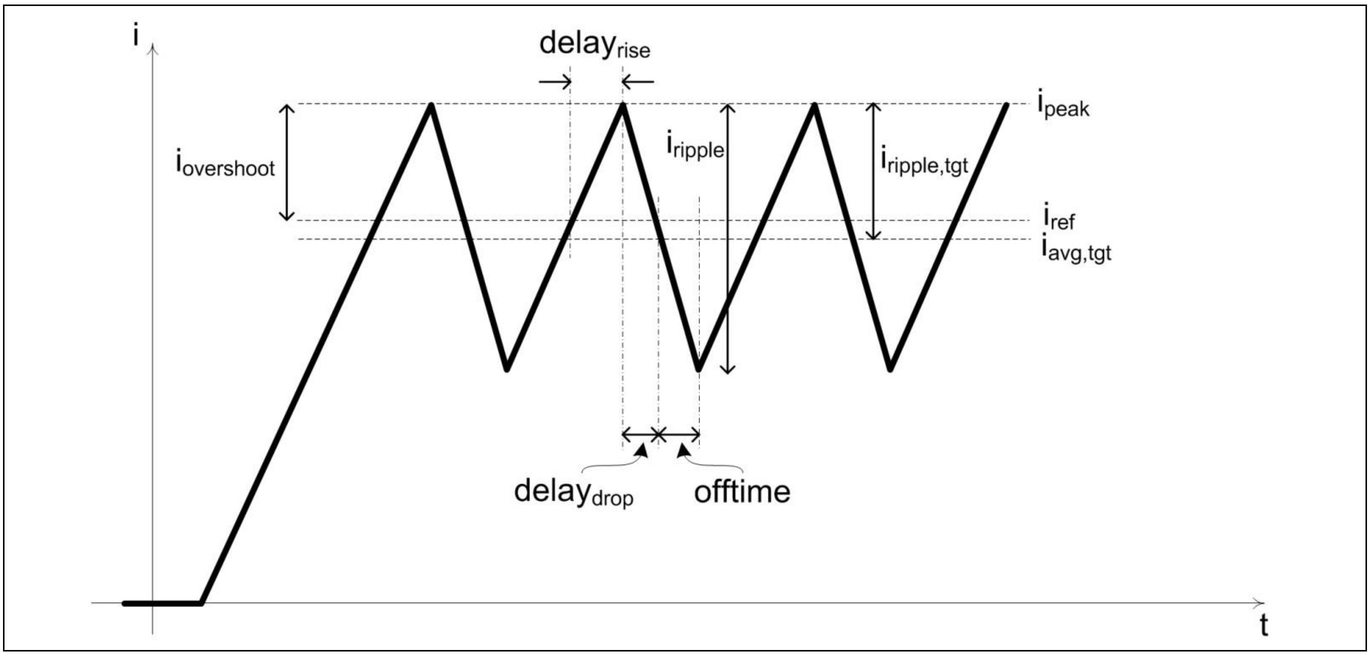

Figure 4 Current ripple tuning parameters

The reference value (i_{ref}) for the peak current level can be set either by an external resistive divider or a pseudo Digital-to-Analog Converter (DAC). The instantaneous LED current will over-shoot the peak reference value slightly (Figure 4), due to internal signal propagation delay in the XMC1000 microcontroller. The amount of over-shoot depends on the propagation delay and the steepness of the LED current rise slope.

The propagation delay is typically a fixed value given for the device and routing option selected. The steepness of the LED current rise slope or how fast the LED current rises depends on V_{in}, LED forward voltage (V_F), and the inductance (L). Typically the inductance is also a fixed value.

Ignoring voltage drops over the diode, shunt resistor, and MOSFET during on-time, the LED current rise slope can be described by the following equation:

$$\begin{aligned}\frac{\delta i}{\delta t} &= \frac{V_{in} - V_F}{L} \\ &= \frac{V_L}{L}\end{aligned}$$

The amount of LED current overshoot is therefore derived as:

$$\begin{aligned}i_{overshoot} &= \frac{\delta i}{\delta t} \times delay_{rise} \\ &= \frac{V_L}{L} \times delay_{rise}\end{aligned}$$

The peak LED current value can then be calculated:

$$i_{peak} = i_{ref} + i_{overshoot}$$

The LED current ripple depends on the period of time the MOSFET is turned off and the steepness of the LED current drop slope. The MOSFET off-time is contributed by a small propagation delay ($delay_{drop}$) and the configured off-time or COMPARE value for the CCU4/8 time slice.

The LED current drop slope can be described as:

$$\begin{aligned}\frac{\delta i}{\delta t} &= \frac{V_F}{L} \\ &= \frac{V_{in} - V_L}{L}\end{aligned}$$

The LED current ripple can then be measured as:

$$\begin{aligned}i_{ripple} &= \frac{\delta i}{\delta t} \times (delay_{drop} + offtime) \\ &= \frac{V_{in} - V_L}{L} \times (delay_{drop} + offtime)\end{aligned}$$

For this automatic current ripple tuning method, the target average LED current value that is to be maintained by the CCM buck LED driver is defined as a value 5% lower than i_{ref}:

$$i_{avg,tgt} = \frac{i_{ref}}{105} \times 100$$

The half-ripple value is defined as:

$$i_{ripple,tgt} = i_{peak} - i_{avg,tgt}$$

To achieve a stable $i_{avg,tgt}$ value, the LED current is expected to drop by a total amount of (2 x $i_{ripple,tgt}$) when the MOSFET is turned off.

The required CCU4/8 off-time can be calculated as:

$$offtime = \frac{L \times 2 \times i_{ripple,tgt}}{V_F} - delay_{drop}$$
$$= \frac{L \times 2 \times i_{ripple,tgt}}{V_{in} - V_L} - delay_{drop}$$

3 Algorithm

The algorithm can be described with the following steps:

1. Measure i_{ref} if external resistive divider is used for the ACMP reference.
2. Calculate $i_{avg,tgt}$.
3. Measure V_{in}.
4. Start MOSFET switching with large ripple configuration.
5. Measure V_L.
6. Calculate V_F.
7. Calculate $i_{overshoot}$.
8. Calculate $i_{ripple,tgt}$.
9. Calculate the required off-time.
10. Update CCU4/8 COMPARE value.
11. Monitor V_{in} and V_F. Repeat tuning process if significant change is observed.

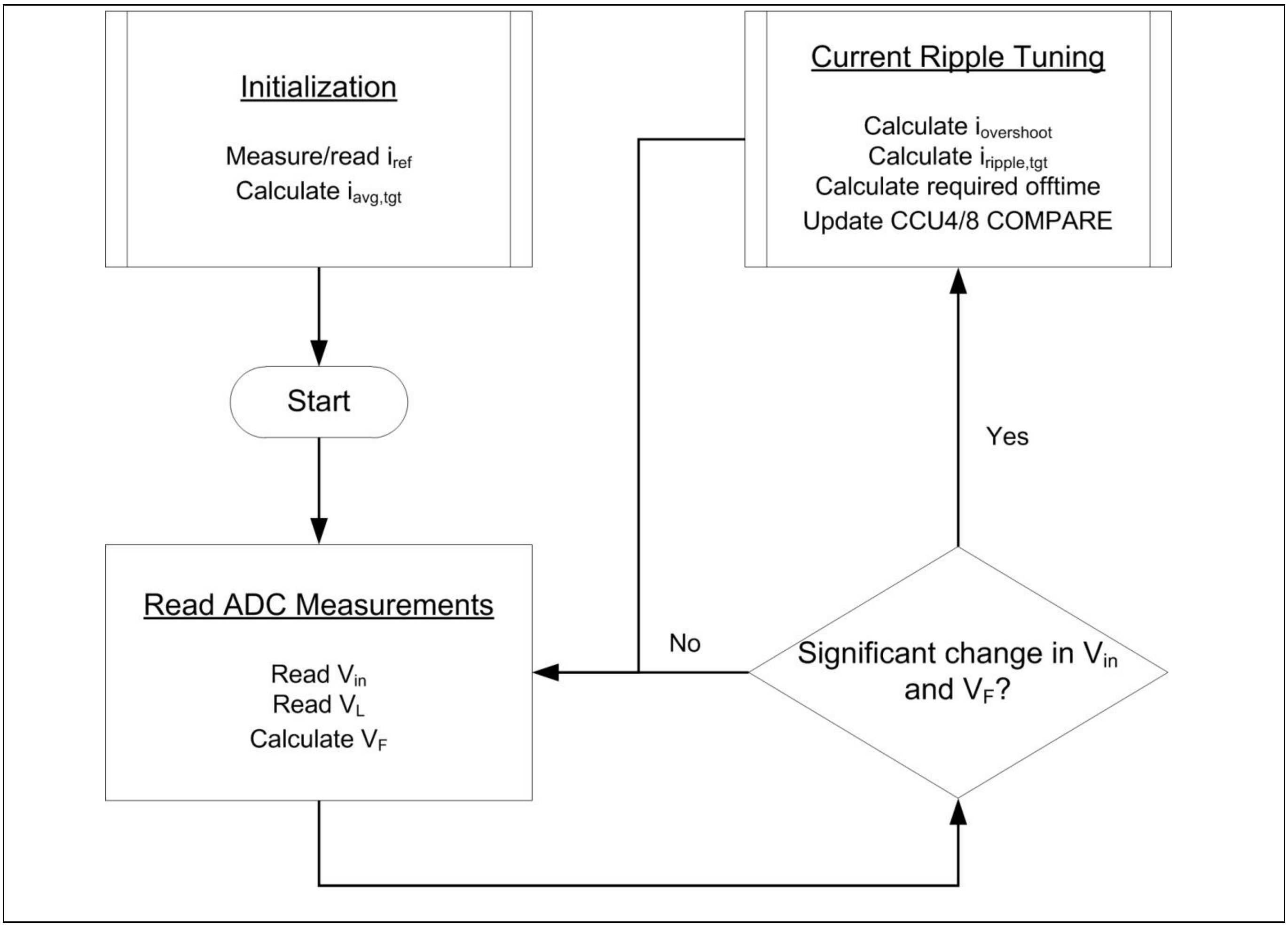

Figure 5 Algorithm flow chart

4 Hardware

The XMC™ LED Current Control Explorer (Figure 6) is an Infineon application kit that supports the automatic LED current ripple tuning method and algorithm.

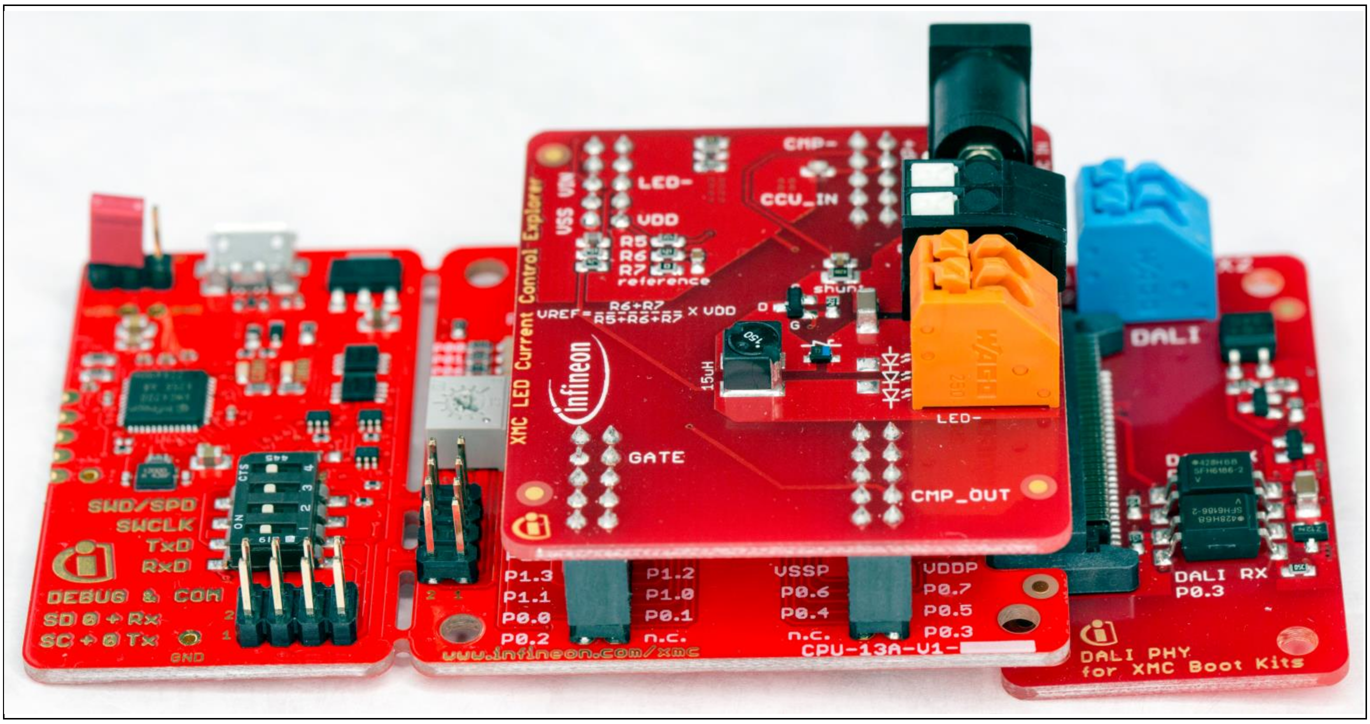

Figure 6 XMC™ LED Current Control Explorer

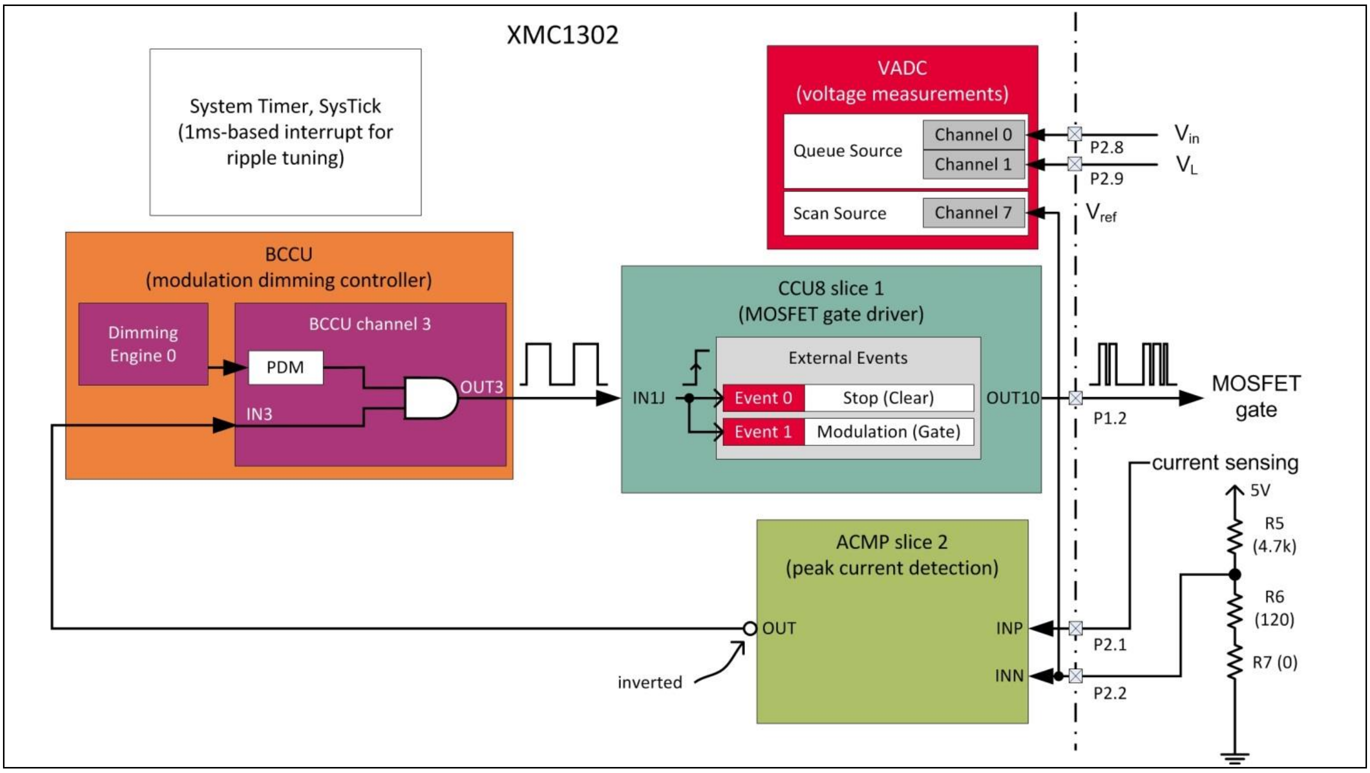

Figure 7 Hardware resources utilized in the XMC™ LED Current Control Explorer kit

4.1 Altering the target average LED current

The kit uses an external resistive divider circuit to generate a reference level for peak current detection. The reference level can be easily altered by using different combinations of R_6 and R_7 resistor values. To assist in the calculations of the required resistor values, an excel workbook titled “Automatic_Current_Ripple_Tuning” is provided. In the sheet “Setup Hardware” (Figure 8), you are able to obtain the required R_6 and R_7 values for the required target average LED current.

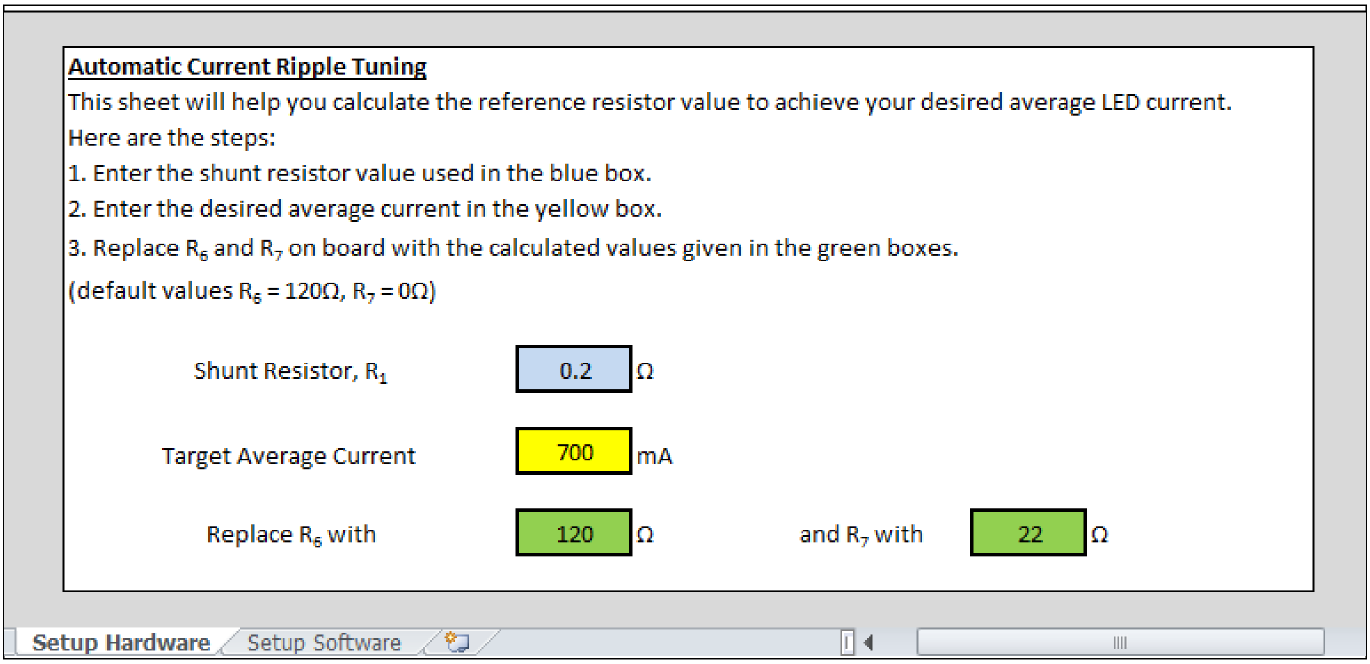

Figure 8 Resistor calculation help sheet

The excel workbook can be downloaded from the same webpage link as this document (Figure 9). Both files can be found in the ‘Documents’ tab (left-hand side of webpage) and under the ‘Application Note’ section (scroll down the webpage).

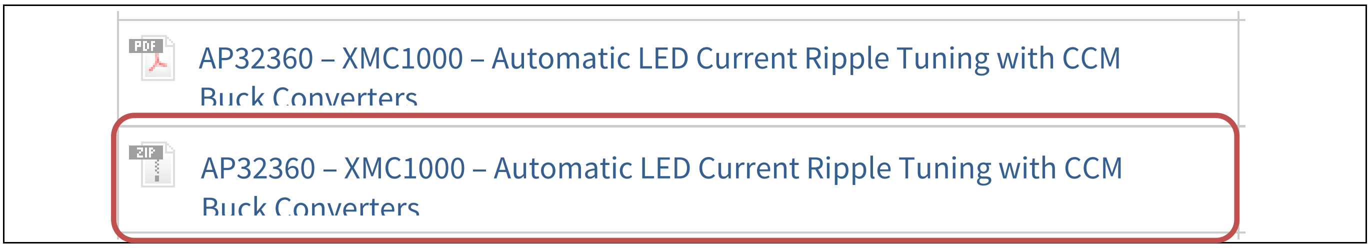

Figure 9 Download link for the excel workbook help-sheet

5 Software

The XMC™ LED Current Control Explorer Kit comes with a project titled "LED_CCEXP_C2_AUTOMATIC_RIPPLE_TUNING" which includes the automatic current ripple tuning feature. The project is developed based on DAVE™ APPs in Infineon's free Integrated Development Environment (IDE) DAVE™. This project can be downloaded from the kit's official webpage.

In the project's main.c file, there is a MACROS section (Figure 10). The values defined have been pre-calculated based on the inductor and shunt resistor values. The scaling factors were selected to avoid overflow cases in the calculation steps.

main.c

```
/* Macros for Automatic Current Ripple Tuning */
#define R_SHUNT                           5U      /* (1/0.2) Ohm */
#define DELAY_DROP                        140U    /* fixed propagation delay when turning off MOSFETs */
#define IREF_CALCULATION_CONSTANT         136533U /* IREF_SCALE_ADJUSTMENT/R_SHUNT/VREF_GAIN*(2^IREF_CALCULATION_SCALING) */
#define IREF_CALCULATION_SCALING          15U
#define IAVG_CALCULATION_CONSTANT         124830U /* 100/105*(2^IAVG_CALCULATION_SCALING) */
#define IAVG_CALCULATION_SCALING          17U
#define IOVERSHOOT_CALCULATION_CONSTANT   1U      /* DELAY_RISE/L*IOVERSHOOT_SCALE_ADJUSTMENT */
#define OFFTIME_CALCULATION_CONSTANT      300U    /* 2*L*OFFTIME_SCALE_ADJUSTMENT */
#define TIMER_COMPARE_CALCULATION_CONSTANT 16777U /* (1/TIMER_RESOLUTION)*(2^TIMER_COMPARE_CALCULATION_SCALING) */
#define TIMER_COMPARE_CALCULATION_SCALING 18U
```

Figure 10 Definition of macros used in the calculations for ripple tuning

The excel workbook "Automatic_Current_Ripple_Tuning" also comes with a worksheet titled "Setup Software" which provides an insight into the calculations of the values used. It also provides the equations described in 2 Method and their actual implementation forms in the code. This help sheet is also useful if you wish to experiment with different inductor and shunt resistor values. It allows you to enter the new values and the respective macro values will be automatically calculated, ready to be copied into the project (Figure 11).

Automatic Current Ripple Tuning

This sheet will help you calculate the values for macros used in the software.

Here are the steps:

1. Enter the inductor value used in the yellow box.
2. Enter the shunt resistor value used in the blue box.
3. Replace values in MACROS section in Main.c with the calculated values in the Macros table (marked in green).

Inductor Value, L: 15 µH

Shunt Resistor, R_1: 0.2 Ω

Macros	
R_SHUNT	5
DELAY_DROP	140
IREF_CALCULATION_CONSTANT	136533
IREF_CALCULATION_SCALING	15
IAVG_CALCULATION_CONSTANT	124830
IAVG_CALCULATION_SCALING	17
IOVERSHOOT_CALCULATION_CONSTANT	1
OFFTIME_CALCULATION_CONSTANT	300
TIMER_COMPARE_CALCULATION_CONSTANT	16777
TIMER_COMPARE_CALCULATION_SCALING	18

Formula
R_SHUNT = 1/R_1
fixed
IREF_CALCULATION_CONSTANT = IREF_SC/
IAVG_CALCULATION_CONSTANT = 100/105
IOVERSHOOT_CALCULATION_CONSTANT =
OFFTIME_CALCULATION_CONSTSANT = 2*l
TIMER_COMPARE_CALCULATION_CONSTA

vare | Setup Software

Figure 11 Help sheet for calculating macro values

Two instances of ADC_MEASUREMENT_ADV APPs are used in the project:

- one to implement the ADC scan source for measuring $V_{ref.}$
- one to implement the ADC queue source for measuring V_{in} and V_L.

The ADC scan source is configured as a one-shot conversion via a software trigger. This trigger is invoked at the start of the program right after the initialization of the peripherals. The result of the conversion is read in the callback function, `void adc_scan_callback(void)`.

The conversions on the ADC queue source are triggered by the Pulse-Density Modulation (PDM) output of the Brightness and Color Control Unit (BCCU) channel. The results of the conversions are read in the callback function, `void adc_queue_callback(void)`. In this callback function the results are checked for significant changes to trigger the current ripple tuning process.

Using the SYSTIMER APP, a software periodic timer with a 1ms-base is created. An interrupt is generated at the end of this timer period, with the tasks defined in the callback function, `void mytimerinterrupt(void)`. Among the tasks carried out in this callback function is the LED current ripple tuning when required feature. The calculations for this task can be found in the function `void current_ripple_tuning(void)`.

Revision history

Major changes since the last revision

Page or Reference	Description of change
All pages.	First release.

Edition 2016-11-04

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32360