Driving LEDs with XMC1000

XMC1000

About this document

Scope and purpose

This document presents different methods for secondary-side driving of LEDs with the XMC1000 microcontroller.

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: AP32275 - Brightness and Color Control Unit (BCCU), http://www.infineon.com/XMC1000

Infineon: AN_201511_PL30_0011 – Pseudo Digital-to-Analog Converter (DAC) with XMC1000, http://www.infineon.com/XMC1000

Table of contents

1 Introduction

Light-emitting diodes (LEDs) are rapidly gaining popularity due to their low cost, energy-efficiency and brightness / color controllability. LED drivers provide an easy (almost plug-and-play) method of driving LEDs. However, we have seen a recent evolution of LED applications; for example, with advancements in the Internet of Things (IoT), LED manufacturers are beginning to integrate communication protocols such as DALI, Bluetooth and Wi-Fi into their lamps. For such applications, flexible microcontrollers can provide a cost-effective implementation.

Infineon's XMC1000 family of 32-bit ARM Cortex-M0 microcontrollers stands out with its Brightness and Color Control Unit (BCCU). The BCCU is a dedicated peripheral for LED dimming control and color mixing using a high-frequency pulse-density modulated (PDM) output to eliminate flicker, even when viewed with a camera. Dimming and color transitions are effectively controlled via hardware such that they appear smooth and natural to the human eye. For more information on the BCCU, please refer to application note AP32275.

In this document, two methods of LED driving will be discussed; i) drive via an external LED driver and ii) drive using a Continuous Conduction Mode (CCM) buck converter.

2 PDM drive via external LED driver

The PDM output of a BCCU channel can be used to drive an LED driver, such as Infineon’s linear LED driver BCR421 or Infineon’s switched-mode LED driver ILD6070. The LED driver will regulate the current supplied to the LED based on the frequency of the input signal that it receives. As the frequency of the PDM output is increased, the LED driver will let more current flow through the LED channel and, as a result, the LED intensity increases.

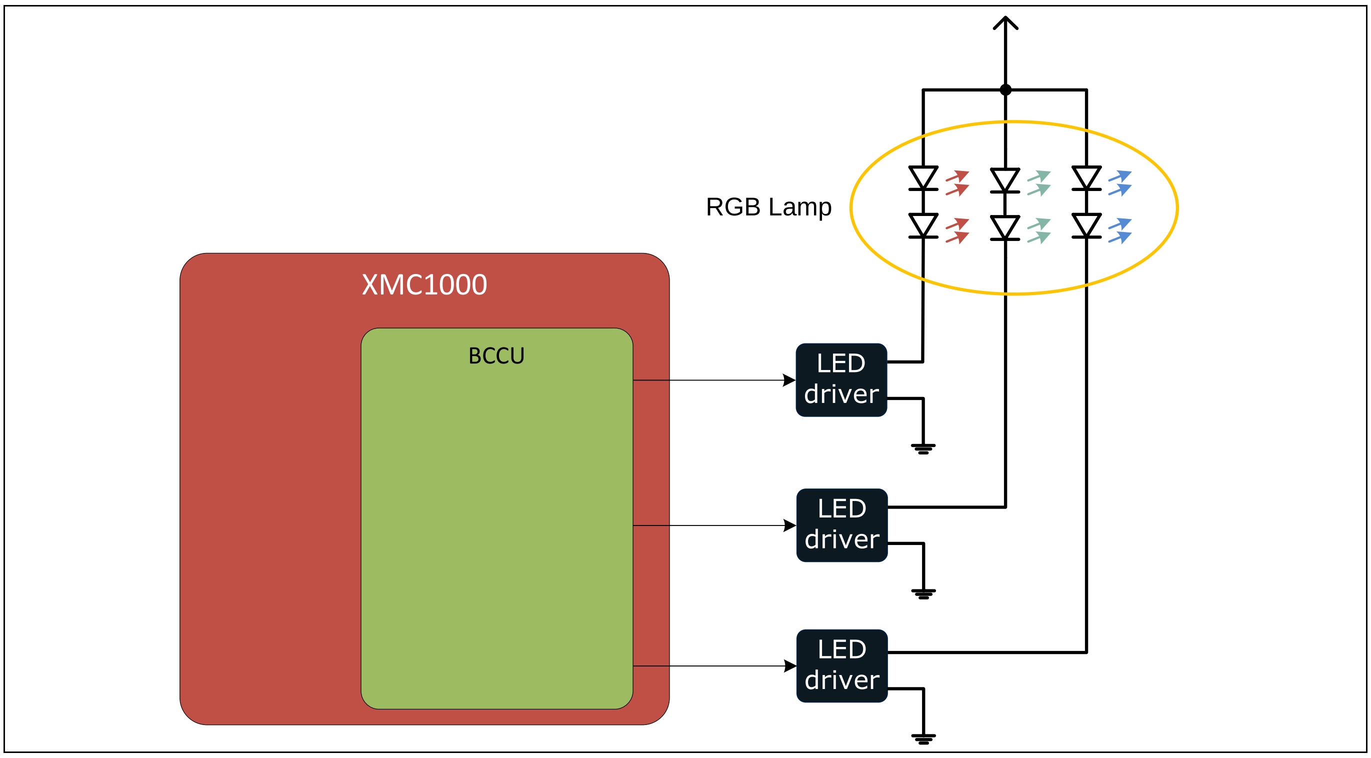


Figure 1 Block diagram – Linear drive for a 3-channel RGB LED lamp

To implement an RGB LED lamp application, three BCCU channels and a BCCU dimming engine are required. Infineon’s LED lighting application Kit (Figure 2) provides an evaluation platform for this method of LED driving. The kit comprises of an XMC1200 boot kit, a color LED card and a white LED card. In Section 2.1, the configurations required for driving one RGB LED lamp are provided.

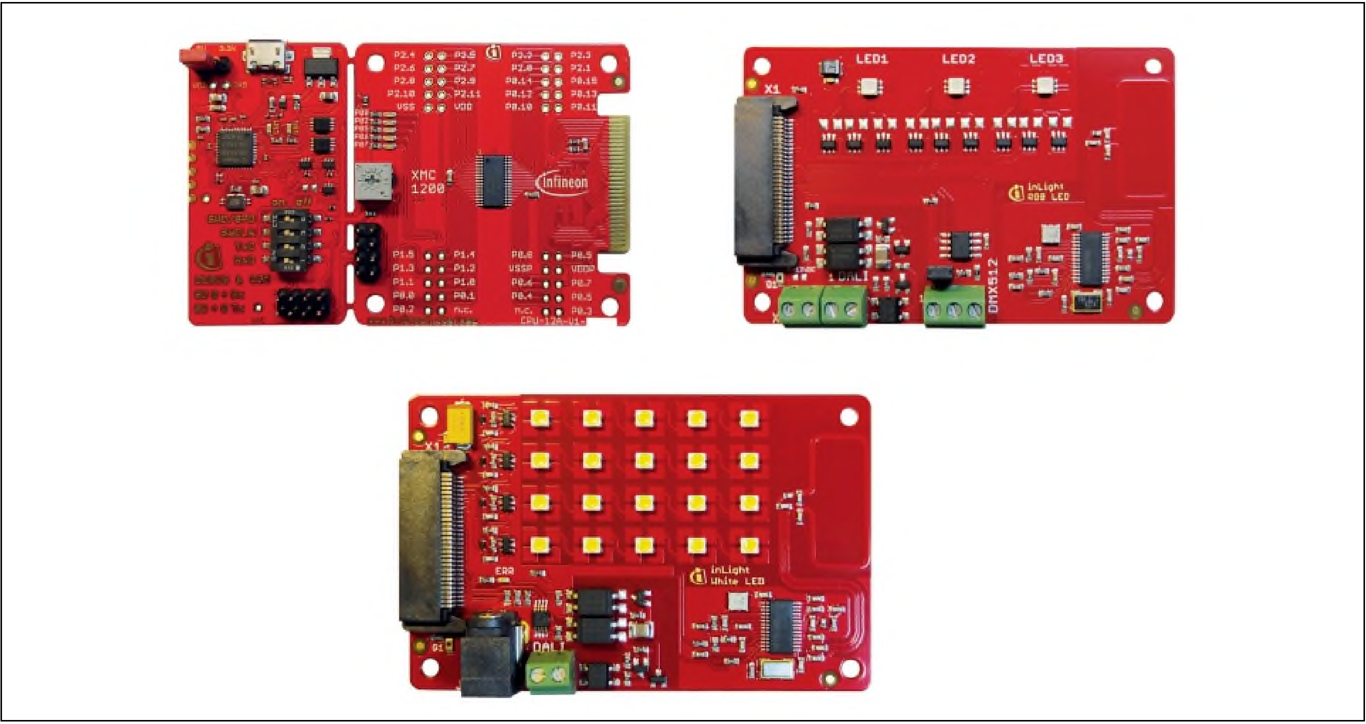


Figure 2 LED lighting application kit

2.1 Example: driving an RGB LED lamp

This section describes how the XMC1200 can be used with Infineon's linear LED driver (BCR421) to drive an RGB LED.

2.1.1 Hardware

This demonstration uses the XMC1200 boot kit and the color LED card from the LED lighting application kit. Only LED1 (Figure 3) will be driven in this example.

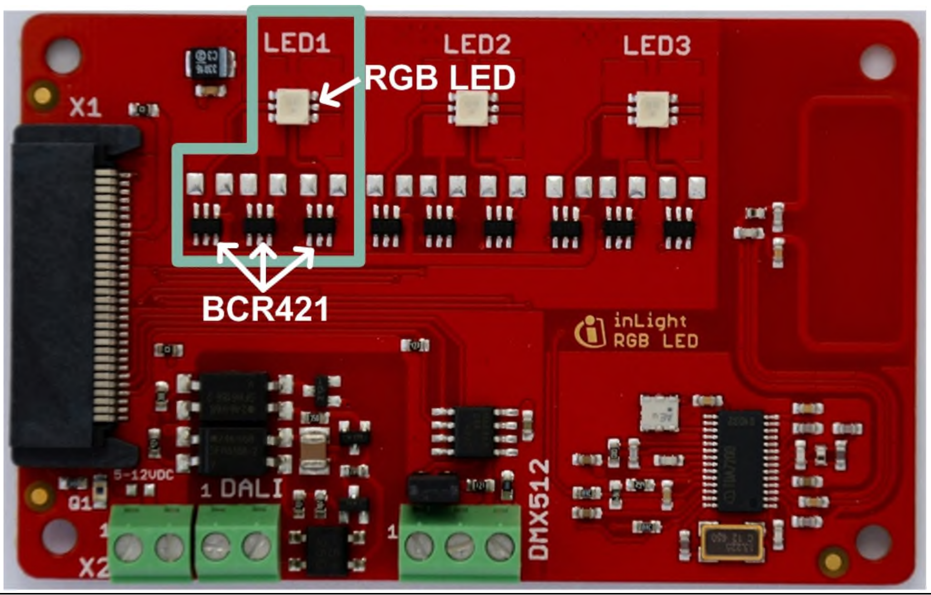


Figure 3 XMC1200 Boot Kit and color LED card

Figure 4 provides an overview of the resources within the XMC1200 that are utilized to drive an RGB LED.

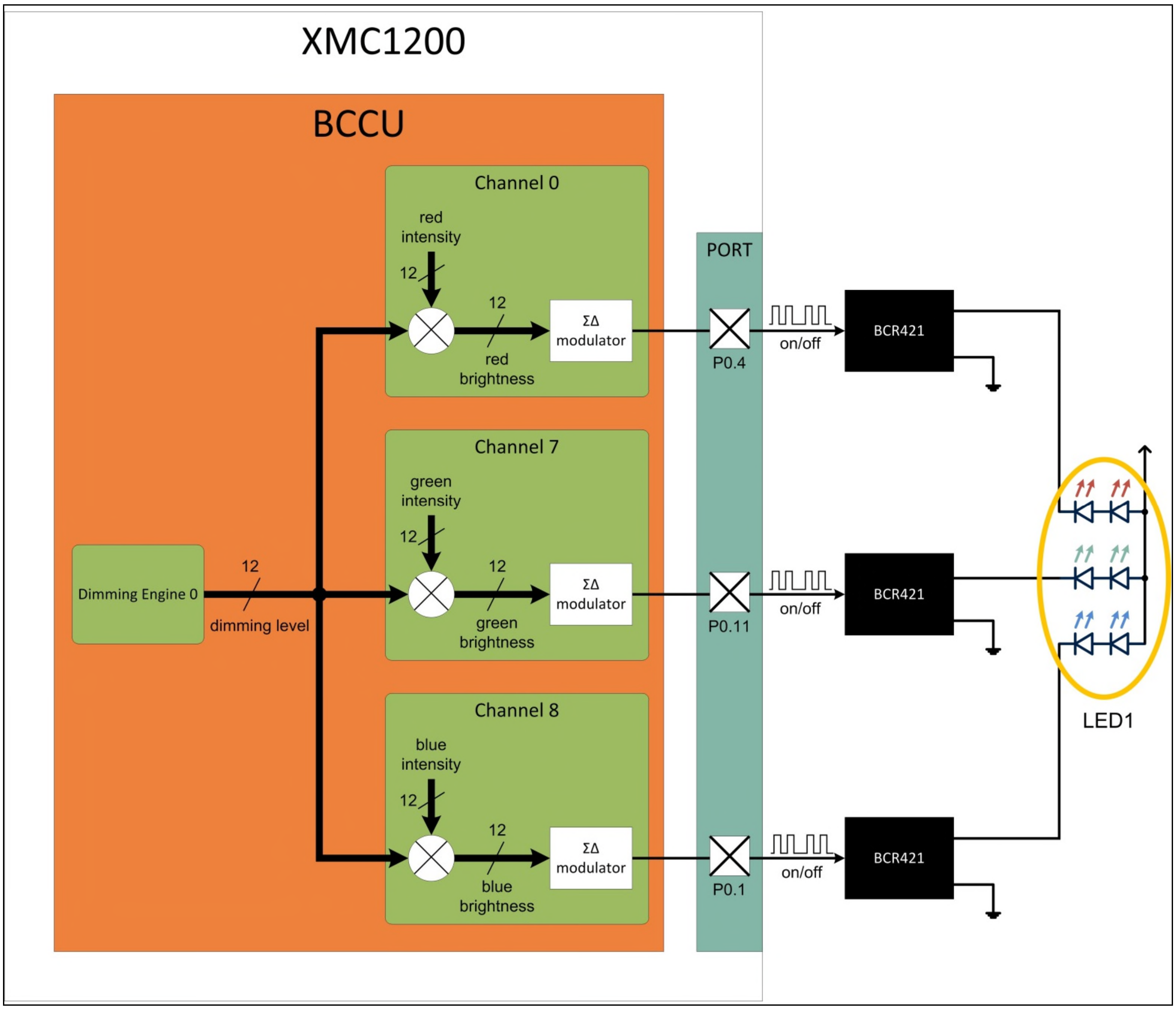


Figure 4 Overview of XMC1200 resources utilized

2.1.2 Software

This section describes the configuration steps for the XMC1200 and is divided into two parts; i) using XMC™ lib and ii) using DAVE™ APPs.

2.1.2.1 Implementation using XMC™ lib

This section describes how the Infineon XMC™ lib can be used to configure the BCCU channels, dimming engine and port pins to drive the RGB LED.

Configuration

The configuration section can be broken down to five parts:

1. System clock configuration:
   The clock configuration utilizes functions and data structures from the SCU library. Therefore, this has to be added to the "Includes" section in the application code.

   ```
   #include <xmc_scu.h>
   ```

   The XMC1200 microcontroller is clocked at 32 MHz. The peripheral clock (PCLK) frequency can be configured to double the value of the main clock (MCLK), which is 64 MHz.

   ```
   XMC_SCU_CLOCK_CONFIG_t clock_config =
   {
     .pclk_src = XMC_SCU_CLOCK_PCLKSRC_DOUBLE_MCLK, /* PCLK=64MHz */
     .fdiv = 0,
     .idiv = 1 /* MCLK=32MHz */
   };
   ```

2. BCCU global configuration:
   This constitutes the configuration of BCCU clocks, which utilizes functions and data structures from the BCCU library. Therefore, this has to be added to the "Includes" section in the application code.

   ```
   #include <xmc_bccu.h>
   ```

   The BCCU fast clock (FCLK) prescaler is configured with a value of 0x50. Assuming that the peripheral clock frequency is configured to 64 MHz, the resulting frequency value is 800 kHz. A BCCU bit-clock (BCLK) frequency that is a quarter of the BCCU FLCK is desired (200 kHz). Therefore XMC_BCCU_BCLK_MODE_NORMAL is configured for the bit-clock selector. The BCCU dimmer clock (DCLK) prescaler is configured with a value of 0xDB to obtain a frequency value of 292.237 kHz. The maximum number of zero-bits at the PDM output is configured to 100 for the flicker watchdog functionality, thus limiting the minimum brightness to 1%.

   ```
   XMC_BCCU_GLOBAL_CONFIG_t bccu_global_config =
   {
     .fclk_ps = 0x50, /* 800kHz @PCLK=64MHz */
     .dclk_ps = 0xdb, /* 292.237kHz @PCLK=64MHz */
     .bclk_sel = XMC_BCCU_BCLK_MODE_NORMAL, /* 200KHz @PCLK=64MHz */
     .maxzero_at_output = 100 /* min. brightness = 1% */
   };
   ```

3. BCCU dimming engine configuration:
   This constitutes the configuration of the fade rate, dither functionality and dimming curve. The fade rate is configured via dim_div, which can be calculated as follows:
   *DIMDIV = (Fade time * DCLK) / 20479* for dimming up
   *DIMDIV = (Fade time * DCLK) / 20734* for dimming down

   As an example, for dimming up with a fade time of 10 seconds:
   DIMDIV = (10 * 292237) / 20479 = 143 or 0x8F

   ```
   XMC_BCCU_DIM_CONFIG_t bccu_dim_config =
   ```

```
{
  .dim_div = 0x8f, /* fade time of 10s */
  .dither_en = 1, /* enable dither */
  .cur_sel = XMC_BCCU_DIM_CURVE_COARSE /* coarse dimming curve */
};
```

4. BCCU channel configuration:

   This constitutes the selection of the dimming input source for each of the three BCCU channels. As the three channels form a single RGB LED, the same dimming engine is selected for all of them. XMC_BCCU_CH_DIMMING_SOURCE_DE0 is used to select dimming engine 0. The flicker watchdog functionality is also enabled for each channel.

```
XMC_BCCU_CH_CONFIG_t r_pdm_config =
{
  .dim_sel = XMC_BCCU_CH_DIMMING_SOURCE_DE0, /* use dimming engine 0 (DE0) */
  .flick_wd_en = 1 /* enable flicker watchdog */
};
XMC_BCCU_CH_CONFIG_t g_pdm_config =
{
  .dim_sel = XMC_BCCU_CH_DIMMING_SOURCE_DE0, /* use dimming engine 0 (DE0) */
  .flick_wd_en = 1 /* enable flicker watchdog */
};
XMC_BCCU_CH_CONFIG_t b_pdm_config =
{
  .dim_sel = XMC_BCCU_CH_DIMMING_SOURCE_DE0, /* use dimming engine 0 (DE0) */
  .flick_wd_en = 1 /* enable flicker watchdog */
};
```

5. GPIO configuration:

   This constitutes the configuration of the port pins for the BCCU channel output, which utilizes functions and data structures from the GPIO library. Therefore, this has to be added to the “Includes” section in the application code.

```
#include <xmc_gpio.h>
```

   BCCU channels 0 and 7 (for red and green channels) use Alternate Output Function 1 (Figure 5) to map their PDM output to P0.4 and P0.11 respectively. Hence, the same configuration data structure can be used for these two channels. For BCCU channel 8, the PDM output is mapped to P0.1 via Alternate Output Function 6.

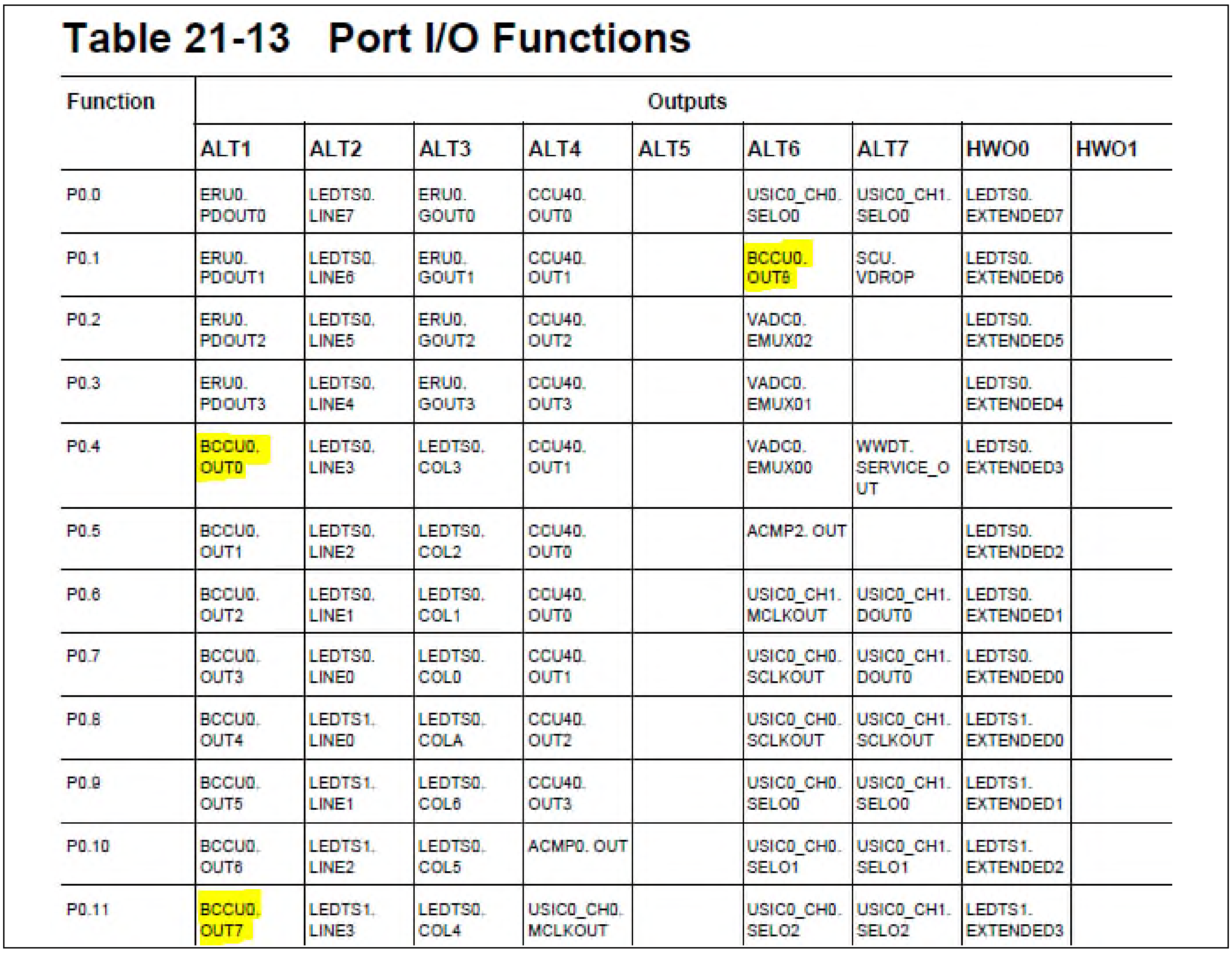


Table 21-13 Port I/O Functions

Function	Outputs								
	ALT1	ALT2	ALT3	ALT4	ALT5	ALT6	ALT7	HWO0	HWO1
P0.0	ERU0. PDOUT0	LEDTS0. LINE7	ERU0. GOUT0	CCU40. OUT0		USIC0_CH0. SELO0	USIC0_CH1. SELO0	LEDTS0. EXTENDED7	
P0.1	ERU0. PDOUT1	LEDTS0. LINE6	ERU0. GOUT1	CCU40. OUT1		BCCU0. OUT8	SCU. VDROP	LEDTS0. EXTENDED6	
P0.2	ERU0. PDOUT2	LEDTS0. LINE5	ERU0. GOUT2	CCU40. OUT2		VADC0. EMUX02		LEDTS0. EXTENDED5	
P0.3	ERU0. PDOUT3	LEDTS0. LINE4	ERU0. GOUT3	CCU40. OUT3		VADC0. EMUX01		LEDTS0. EXTENDED4	
P0.4	BCCU0. OUT0	LEDTS0. LINE3	LEDTS0. COL3	CCU40. OUT1		VADC0. EMUX00	WWDT. SERVICE_O UT	LEDTS0. EXTENDED3	
P0.5	BCCU0. OUT1	LEDTS0. LINE2	LEDTS0. COL2	CCU40. OUT0		ACMP2. OUT		LEDTS0. EXTENDED2	
P0.6	BCCU0. OUT2	LEDTS0. LINE1	LEDTS0. COL1	CCU40. OUT0		USIC0_CH1. MCLKOUT	USIC0_CH1. DOUT0	LEDTS0. EXTENDED1	
P0.7	BCCU0. OUT3	LEDTS0. LINE0	LEDTS0. COL0	CCU40. OUT1		USIC0_CH0. SCLKOUT	USIC0_CH1. DOUT0	LEDTS0. EXTENDED0	
P0.8	BCCU0. OUT4	LEDTS1. LINE0	LEDTS0. COLA	CCU40. OUT2		USIC0_CH0. SCLKOUT	USIC0_CH1. SCLKOUT	LEDTS1. EXTENDED0	
P0.9	BCCU0. OUT5	LEDTS1. LINE1	LEDTS0. COL6	CCU40. OUT3		USIC0_CH0. SELO0	USIC0_CH1. SELO0	LEDTS1. EXTENDED1	
P0.10	BCCU0. OUT6	LEDTS1. LINE2	LEDTS0. COL5	ACMP0. OUT		USIC0_CH0. SELO1	USIC0_CH1. SELO1	LEDTS1. EXTENDED2	
P0.11	BCCU0. OUT7	LEDTS1. LINE3	LEDTS0. COL4	USIC0_CH0. MCLKOUT		USIC0_CH0. SELO2	USIC0_CH1. SELO2	LEDTS1. EXTENDED3	

Figure 5 Snapshot of Port I/O functions table from reference manual

```
XMC_GPIO_CONFIG_t r_g_pdm_output_config =
{
    .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT1
};

XMC_GPIO_CONFIG_t b_pdm_output_config =
{
    .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT6
};
```

The unused LED pins also have to be configured to a known state to prevent them from illuminating.

```
XMC_GPIO_CONFIG_t unused_led_output_config =
{
   .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL,
   .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW
};
```

Initialization

Similarly, the initialization can be broken down to five parts:

1. System clock Initialization:
   This step initializes the MCLK to 32 MHz and PCLK to 64 MHz respectively.
   ```
   /* Ensure peripheral clock frequency is set at 64MHz (2*MCLK) */
   XMC_SCU_CLOCK_Init(&clock_config);
   ```

2. BCCU global initialization:
   It is essential that the BCCU global functions are initialized first. This also ungates the peripheral clock to the BCCU kernel.
   ```
   /* BCCU Global Initialization */
   XMC_BCCU_GlobalInit(BCCU0, &bccu_global_config);
   ```

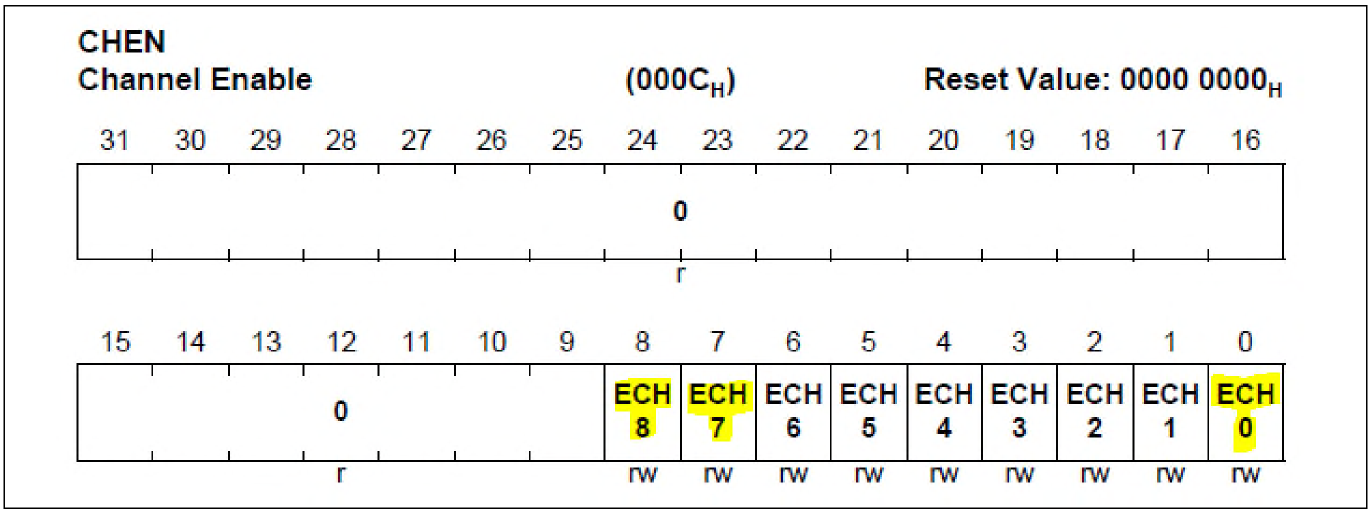
3. Dimming engine intialization:
   ```
   /* init dimming engine */
   XMC_BCCU_DIM_Init(BCCU0_DE0, &bccu_dim_config);

   /* enable dimming engine */
   XMC_BCCU_EnableDimmingEngine(BCCU0, XMC_BCCU_CH_DIMMING_SOURCE_DE0);

   /* set target dimming level to max */
   XMC_BCCU_DIM_SetTargetDimmingLevel(BCCU0_DE0, 4095);
   ```
   After the target dimming level is set, the change in dimming level can be effected by starting the dimming process.
   ```
   /* effect the change in dimming level */
   XMC_BCCU_StartDimming(BCCU0, XMC_BCCU_CH_DIMMING_SOURCE_DE0);
   ```

4. BCCU channel Initialization:
   The three BCCU channels that are used are BCCU0_CH0, BCCU0_CH7 and BCCU0_CH8 for the red, green and blue LED channels respectively.
   ```
   /* init channels */
   XMC_BCCU_CH_Init(BCCU0_CH0, &r_pdm_config);
   XMC_BCCU_CH_Init(BCCU0_CH7, &g_pdm_config);
   XMC_BCCU_CH_Init(BCCU0_CH8, &b_pdm_config);
   ```
   The linear walk time is set to an initial value of 7 seconds, by setting the prescaler to a value of 0x2AC. The prescaler can be calculated using this formula:

   *Linear prescaler = (Linear walk * FCLK) / 2^13*
   ```
   /* Set linear walk time for PDMs*/
   ```

```
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH0, 0x2AC);
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH7, 0x2AC);
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH8, 0x2AC);
```

The channels are then enabled concurrently. The mask required to enable channels 0, 7 and 8 is 0x181 (Figure 6).



CHEN
Channel Enable (000C$_H$) Reset Value: 0000 0000$_H$

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
0															
r															

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0							ECH 8	ECH 7	ECH 6	ECH 5	ECH 4	ECH 3	ECH 2	ECH 1	ECH 0
r							rw	rw	rw	rw	rw	rw	rw	rw	rw

Figure 6 Snapshot of BCCU channel enable (CHEN) register

```
/* Enable PDM channels*/
XMC_BCCU_ConcurrentEnableChannels(BCCU0, 0x181);
```

The color of the RGB LED can then be initialized by setting the target channel intensities. To set the LED color to white, set each target channel intensity with the same value.

```
/* set initial color to white */
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 1365);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH7, 1365);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH8, 1365);
```

To effect the color change, start the linear walk. For a smooth color transition, start the linear walk concurrently in all three channels. The same mask can be used.

```
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x181);
```

5. GPIO Initialization:

```
/* Enable BCCU PDM output */
XMC_GPIO_Init(P0_4, &r_g_pdm_output_config);
XMC_GPIO_Init(P0_11, &r_g_pdm_output_config);
XMC_GPIO_Init(P0_1, &b_pdm_output_config);

/* Configure and turn off unused LEDs */
XMC_GPIO_Init(P0_5, &unused_led_output_config);
XMC_GPIO_Init(P0_6, &unused_led_output_config);
XMC_GPIO_Init(P0_7, &unused_led_output_config);
XMC_GPIO_Init(P0_8, &unused_led_output_config);
XMC_GPIO_Init(P0_9, &unused_led_output_config);
XMC_GPIO_Init(P0_10, &unused_led_output_config);
```

Application code

This section provides a guide to adjust the dimming level and color of the LED lamp in-application.

1. Dimming level

   To change the lamp brightness, the target dimming level must be set first:

   ```
   /* set target dimming level to 50% brightness */
   XMC_BCCU_DIM_SetTargetDimmingLevel(BCCU0_DE0, 2048);
   ```

   After the target dimming level is set, the change in dimming level can be effected by starting the dimming process:

   ```
   /* start dimming process */
   XMC_BCCU_StartDimming(BCCU0, XMC_BCCU_CH_DIMMING_SOURCE_DE0);
   ```

   To change the fade time, use the following function before starting the dimming process:

   ```
   /* set fade time to 3s */
   XMC_BCCU_DIM_SetDimDivider(BCCU0_DE0, 0x2B);
   ```

2. Color

   To change the color of the LED, first set the target channel intensities. The combination of the channel intensities results in a color. One simple color scheme that can be suggested is as follows (Table 1), whereby the respective channel intensities sum up to a maximum intensity value of 4095 or 4096.

Table 1 Simple color scheme

Desired color	RED channel intensity	GREEN channel intensity	BLUE channel intensity
Red	4095	0	0
Green	0	4095	0
Blue	0	0	4095
White	1365	1365	1365
Yellow	2048	2048	0
Magenta	2048	0	2048
Cyan	0	2048	2048

For example, to change the color of the LED to red:

```
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 4095);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH7, 0);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH8, 0);
```

Start the linear walk concurrently in all channels.

```
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x181);
```

2.1.2.2 Implementation using DAVE™ APP

This section describes how the PDM_DIMMED_LED_LAMP APP in DAVE™ (Figure 7) can be used to configure the BCCU channels, dimming engine and port pins to drive the RGB LED.

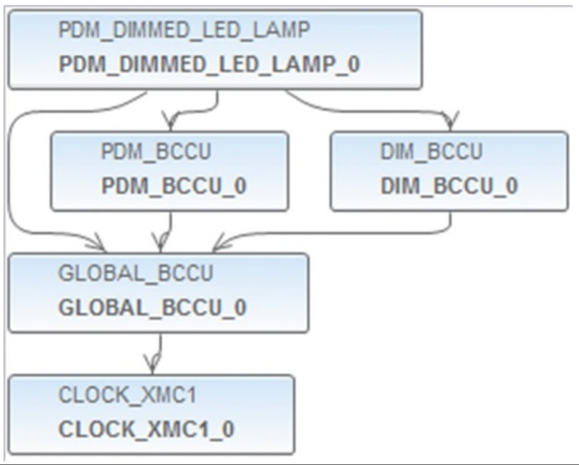
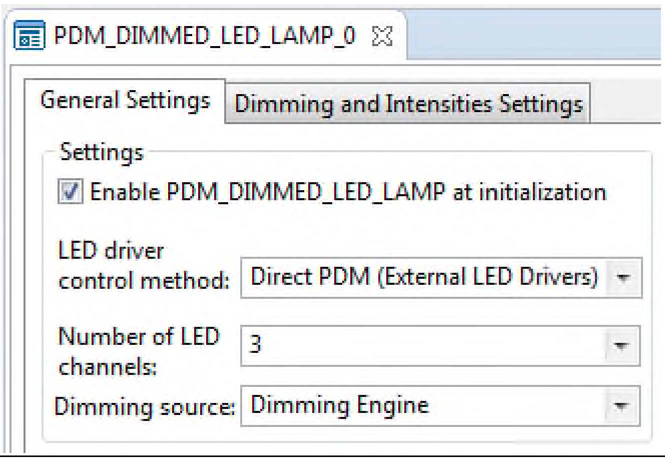


Figure 7 PDM_DIMMED_LED_LAMP APP in DAVE™

The PDM_DIMMED_LED_LAMP APP UI provides configurations for the LED driver control method, number of LED channels and dimming source under the "General Settings" tab (Figure 8), initial dimming level, channel intensities, linear walk time and fade rate under the "Dimming and Intensities Settings" tab (Figure 9).


PDM_DIMMED_LED_LAMP_0

General Settings | Dimming and Intensities Settings

Settings

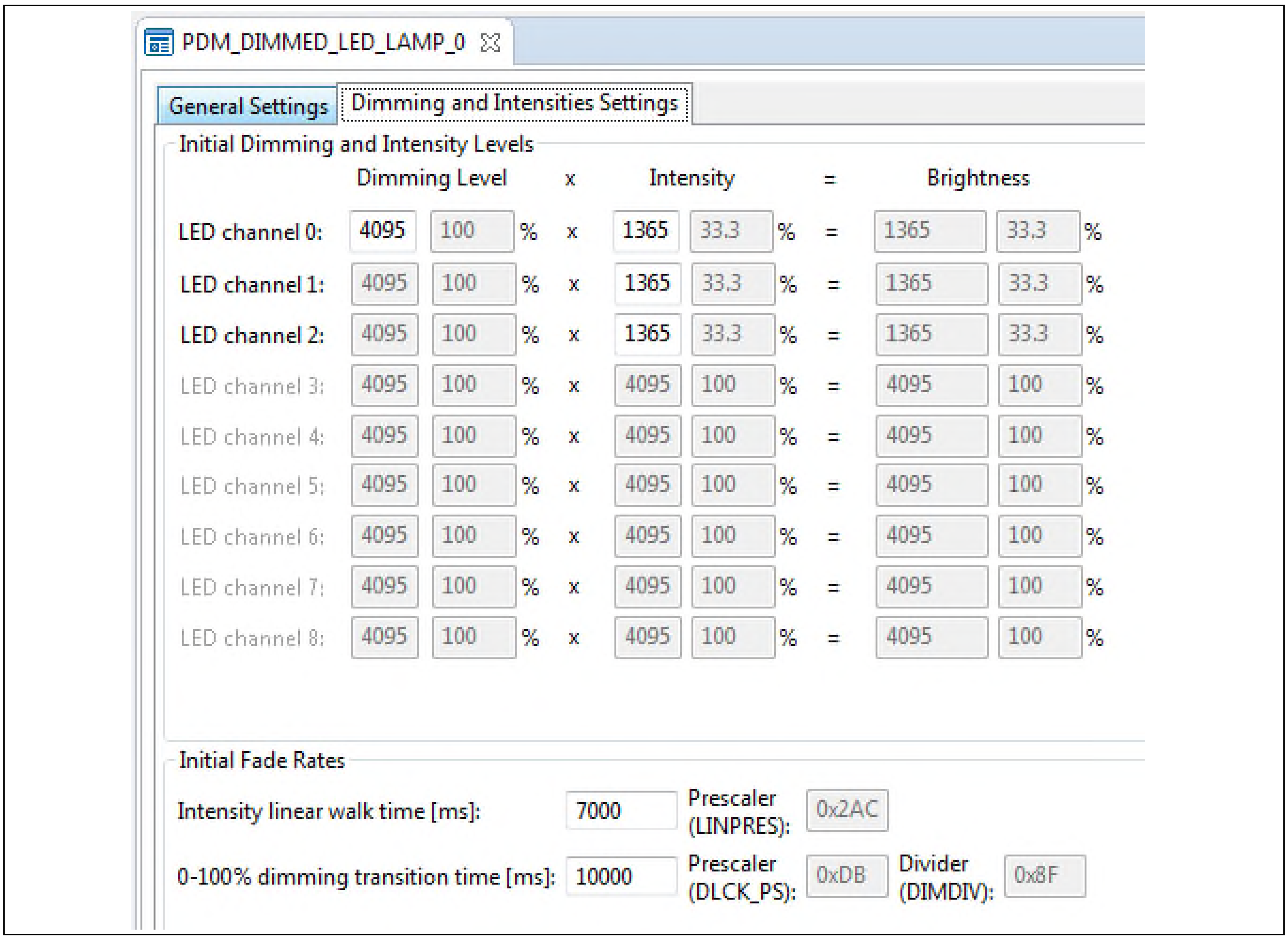
[✓] Enable PDM_DIMMED_LED_LAMP at initialization

LED driver control method: Direct PDM (External LED Drivers)

Number of LED channels: 3

Dimming source: Dimming Engine

Figure 8 PDM_DIMMED_LED_LAMP APP configurations (General Settings tab)



PDM_DIMMED_LED_LAMP_0

General Settings | Dimming and Intensities Settings

Initial Dimming and Intensity Levels

	Dimming Level			x	Intensity			=	Brightness		
LED channel 0:	4095	100	%	x	1365	33.3	%	=	1365	33.3	%
LED channel 1:	4095	100	%	x	1365	33.3	%	=	1365	33.3	%
LED channel 2:	4095	100	%	x	1365	33.3	%	=	1365	33.3	%
LED channel 3:	4095	100	%	x	4095	100	%	=	4095	100	%
LED channel 4:	4095	100	%	x	4095	100	%	=	4095	100	%
LED channel 5:	4095	100	%	x	4095	100	%	=	4095	100	%
LED channel 6:	4095	100	%	x	4095	100	%	=	4095	100	%
LED channel 7:	4095	100	%	x	4095	100	%	=	4095	100	%
LED channel 8:	4095	100	%	x	4095	100	%	=	4095	100	%

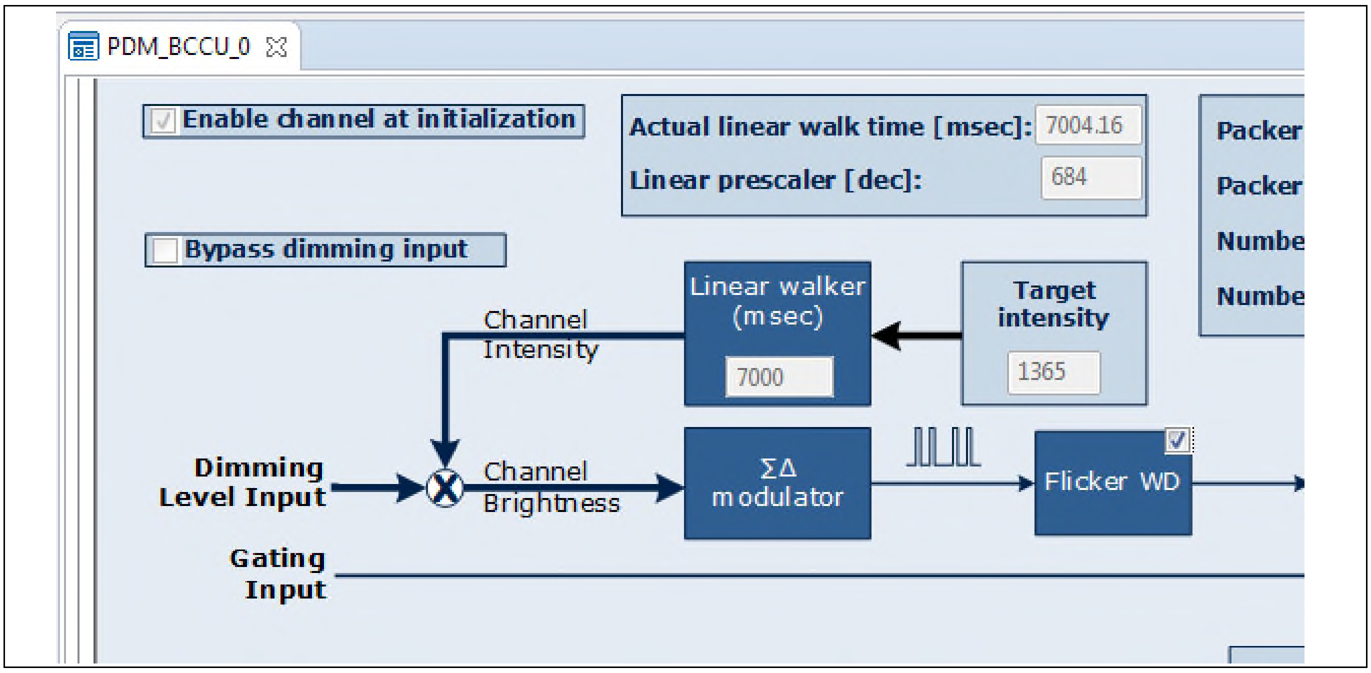
Initial Fade Rates

Intensity linear walk time [ms]: 7000 Prescaler (LINPRES): 0x2AC

0-100% dimming transition time [ms]: 10000 Prescaler (DLCK_PS): 0xDB Divider (DIMDIV): 0x8F

Figure 9 PDM_DIMMED_LED_LAMP_APP configurations (Dimming and Intensities Settings tab)

The PDM_BCCU APP UI provides configurations for the BCCU channel such as flicker watchdog, packer, gate and trigger functionalities. Enable the flicker watchdog functionality by checking the box (Figure 10) for all 3 instances of the PDM_BCCU APPs.


PDM_BCCU_0

☑ Enable channel at initialization

Actual linear walk time [msec]: 7004.16

Linear prescaler [dec]: 684

☐ Bypass dimming input

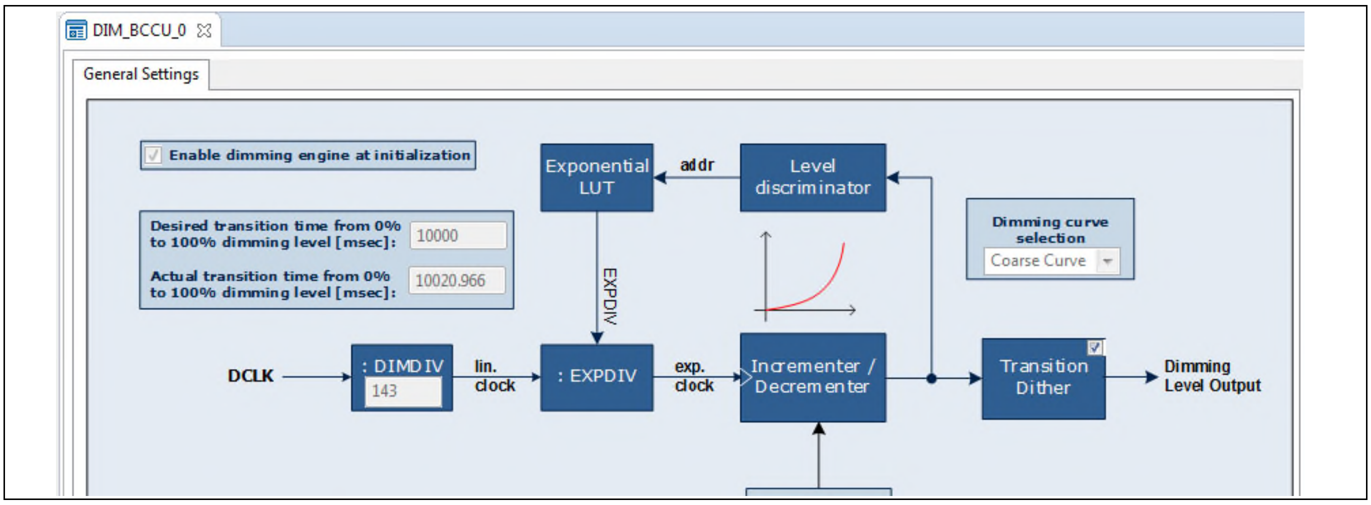
Linear walker (msec): 7000

Target intensity: 1365

☑ Flicker WD

Figure 10 PDM_BCCU APP configuration

The DIM_BCCU APP UI provides configurations for the dimming engine such as dimming curve and dither functionality. Enable the dither functionality by checking the box (Figure 11).


DIM_BCCU_0

General Settings

☑ Enable dimming engine at initialization

Desired transition time from 0% to 100% dimming level [msec]: 10000

Actual transition time from 0% to 100% dimming level [msec]: 10020.966

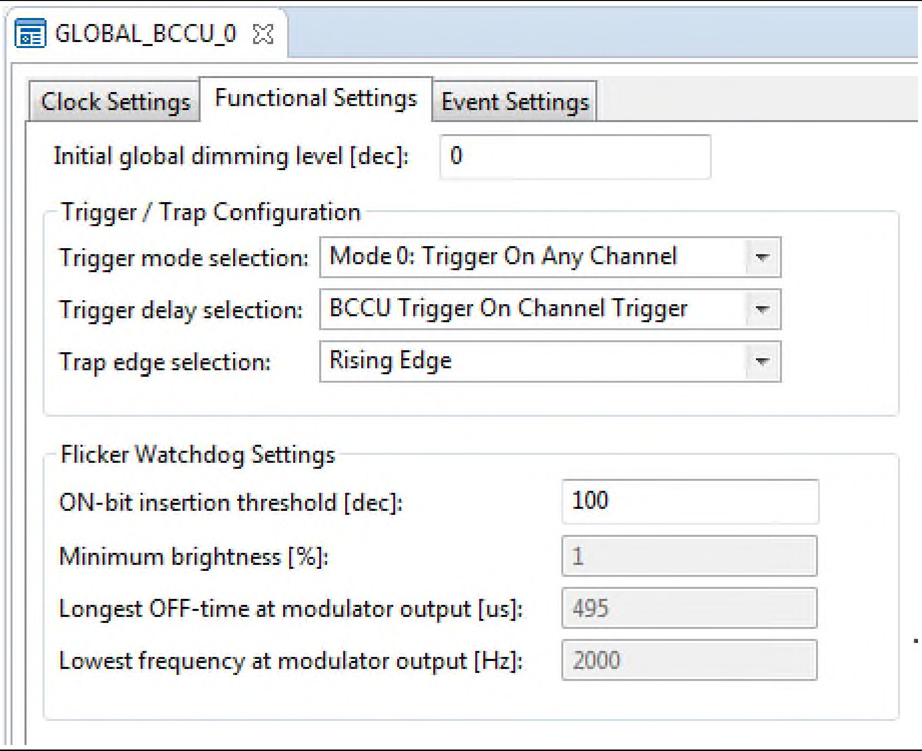
Dimming curve selection: Coarse Curve

: DIMDIV 143

☑ Transition Dither

Figure 11 DIM_BCCU APP configuration

The GLOBAL_BCCU APP UI provides configurations for the BCCU at a global level such as respective clock frequencies, flicker watchdog settings and interrupt event settings. Under the “Functional Settings” tab, configure the ON-bit insertion threshold to 100 (Figure 12).



Initial global dimming level [dec]: 0

Trigger / Trap Configuration

Trigger mode selection: Mode 0: Trigger On Any Channel

Trigger delay selection: BCCU Trigger On Channel Trigger

Trap edge selection: Rising Edge

Flicker Watchdog Settings

ON-bit insertion threshold [dec]: 100

Minimum brightness [%]: 1

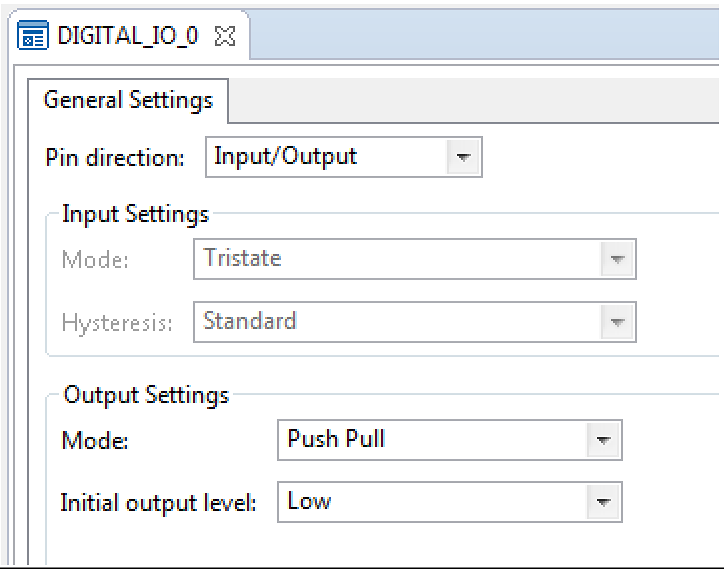
Longest OFF-time at modulator output [us]: 495

Lowest frequency at modulator output [Hz]: 2000

Figure 12 GLOBAL_BCCU APP configuration (Functional Settings tab)

The CLOCK_XMC1 APP UI provides configurations for the system clock. The PCLK has been configured to 64 MHz by default, thus no alteration is needed.

For the unused LED pins that are connected to LED2 and LED3, the DIGITAL_IO APP is used to configure these pins to a known state (Figure 13). Six instances of the DIGITAL_IO APP are needed to configure the six unused pins.



Pin direction: Input/Output

Input Settings

Mode: Tristate

Hysteresis: Standard

Output Settings

Mode: Push Pull

Initial output level: Low

Figure 13 DIGITAL_IO APP configuration for unused LED pins

Pin assignment

Ideally, for an RGB LED, the RED, GREEN and BLUE LEDs are assigned to BCCU channels in a sequential manner to facilitate in-application control. To make a sequential channel assignment:

1. Hover the mouse cursor over the **connecting arrow** to a PDM_BCCU APP. A label appears momentarily showing the LED number, for example: LED0 (Figure 14).

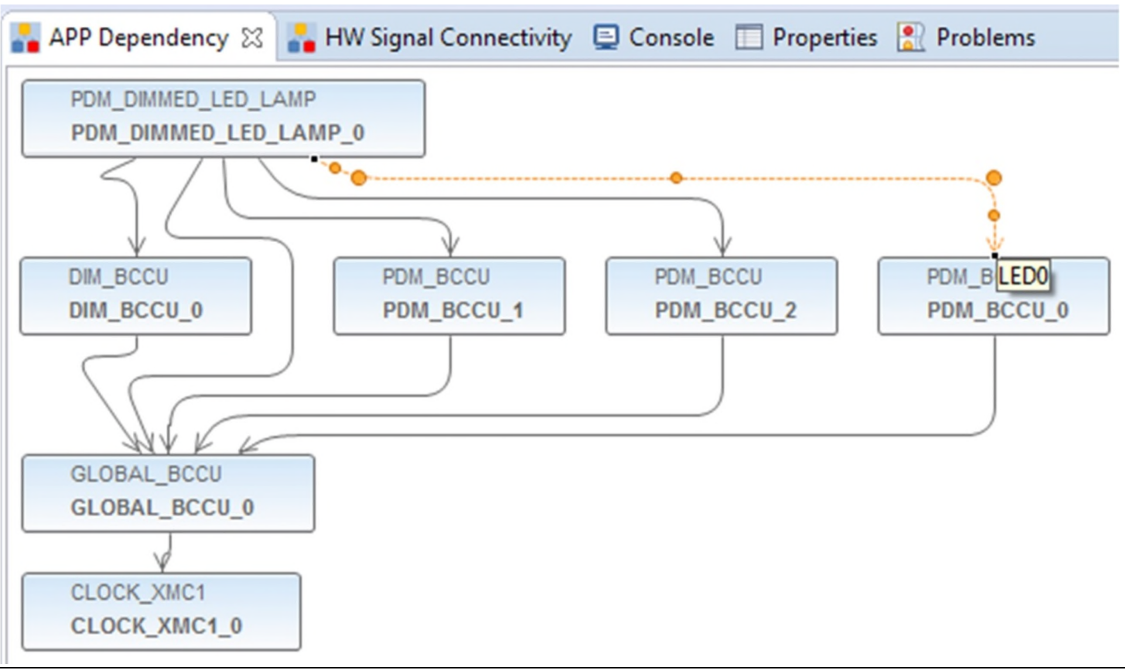
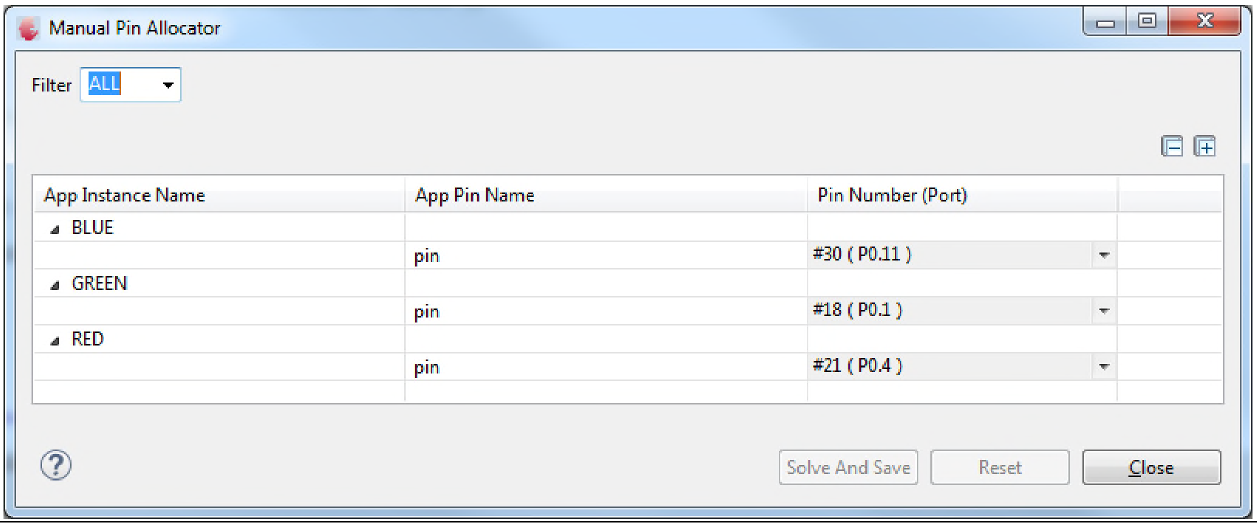


Figure 14 Renaming PDM_BCCU instance label

2. Right-click the PDM_BCCU APP.
3. Select “Rename instance label”.
4. Type in new instance label. Replace the label of the instance “LED0” with “RED”.
5. Repeat steps 1 to 4 for the other PDM_BCCU APP instances. Replace “LED1” with “GREEN” and “LED2” with “BLUE”.
6. Manually assign pins to the renamed PDM_BCCU APP instances (Figure 15).

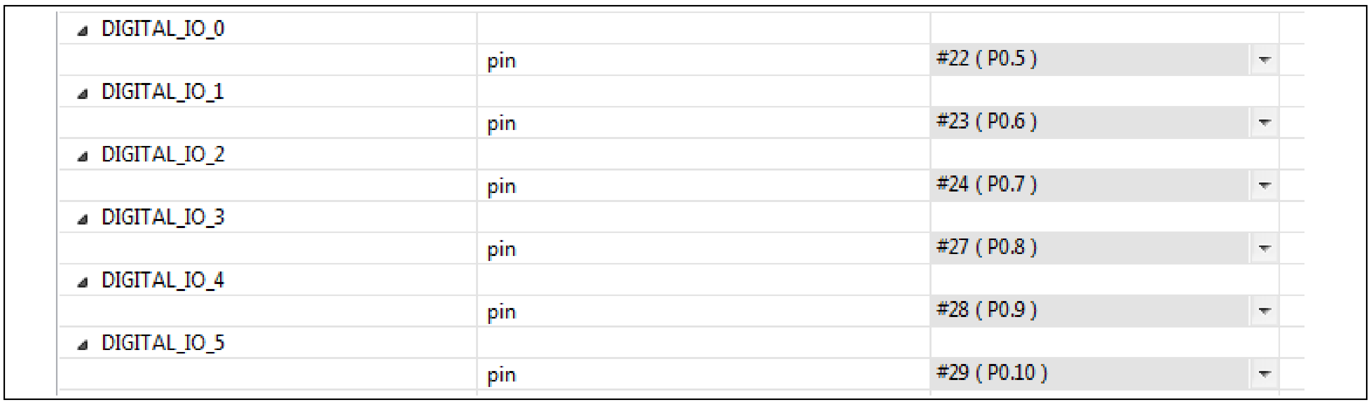


Filter: ALL

App Instance Name	App Pin Name	Pin Number (Port)
BLUE		
	pin	#30 (P0.11)
GREEN		
	pin	#18 (P0.1)
RED		
	pin	#21 (P0.4)

Figure 15 Manual pin assignment for PDM_BCCU APPs

For the unused LED pins, the pin assignment is shown in Figure 16.



DIGITAL_IO_0		
	pin	#22 (P0.5)
DIGITAL_IO_1		
	pin	#23 (P0.6)
DIGITAL_IO_2		
	pin	#24 (P0.7)
DIGITAL_IO_3		
	pin	#27 (P0.8)
DIGITAL_IO_4		
	pin	#28 (P0.9)
DIGITAL_IO_5		
	pin	#29 (P0.10)

Figure 16 Manual pin assignment for unused LED pins

Application code

This section provides a guide to adjust the dimming level and color of the LED lamp in-application.

1. Dimming level

   The DAVE™-generated configuration data structure, *PDM_DIMMED_LED_LAMP_0.config*, for the PDM_DIMMED_LED_LAMP_0 APP instance also holds the dim level value of the dimming engine used in the lamp. It acts as a shadow register for the respective register bit field **DLSz.TDLEV**.

   As an example, to change the dim level to 100%:

```
PDM_DIMMED_LED_LAMP_0_config.dim_level = 4095;
```

   The line above sets the dimming engine target level for the desired dim level. To effect the brightness change, the dimming process has to be started:

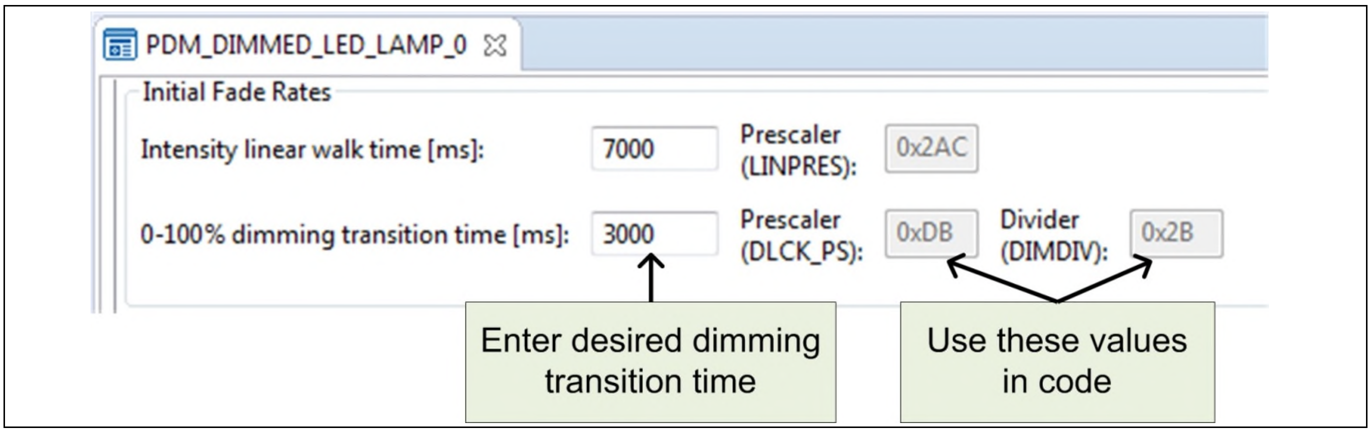
```
PDM_DIMMED_LED_LAMP_SetDimLevelExponential(&PDM_DIMMED_LED_LAMP_0);
```

   This function programs the BCCU register bit fields with the configured dimming transition time and the target dimming level and also starts the dimming process. The dimming transition time can be configured the same way as previously in the UI of the PDM_DIMMED_LED_LAMP APP.

   To use a different dimming transition time for changing lamp brightness:

```
/* Configure dimming transition time = 3s, and start dimming process */
PDM_DIMMED_LED_LAMP_SetDimLevelExponentialAdv(&PDM_DIMMED_LED_LAMP_0, 0x2B, 0xDB);
```

   The PDM_DIMMED_LED_LAMP APP UI can be used to easily calculate the prescaler and divider values for the desired dimming transition time (Figure 17).



PDM_DIMMED_LED_LAMP_0

Initial Fade Rates

Intensity linear walk time [ms]: 7000 Prescaler (LINPRES): 0x2AC

0-100% dimming transition time [ms]: 3000 Prescaler (DLCK_PS): 0xDB Divider (DIMDIV): 0x2B

Figure 17 Use PDM_DIMMED_LED_LAMP APP to calculate dimming transition time prescaler and divider

2. Color

   PDM_DIMMED_LED_LAMP_0.config also holds the intensity values of the respective BCCU channels used in the lamp and also acts like a shadow register for the respective register bit fields INTSy.TCHINT.

   As an example, to change the light color to red:

```
PDM_DIMMED_LED_LAMP_0.config->led_intensity[0] = 4095;
PDM_DIMMED_LED_LAMP_0.config->led_intensity[1] = 0;
PDM_DIMMED_LED_LAMP_0.config->led_intensity[2] = 0;
```

   The lines above set the respective channel intensities for the desired color. To effect the color change, the linear walk has to be started:

```
PDM_DIMMED_LED_LAMP_SetColor(&PDM_DIMMED_LED_LAMP_0);
```

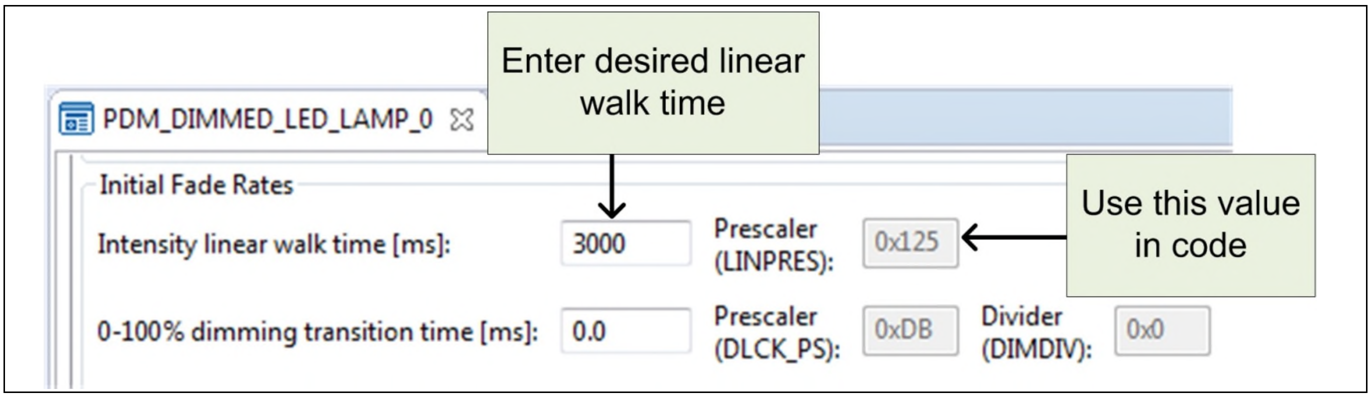
   This function programs the BCCU register bit fields with the requested intensity values and then synchronizes the start of the linear walk for all channels of the lamp, to ensure a smooth and natural color transition. The linear walk time is configured the same as previously in the UI of the PDM_DIMMED_LED_LAMP APP.

   The combination of the channel intensities results in a color. One simple color scheme that can be followed is as suggested in Table 1.

   To use a different linear walk time for changing lamp color:

```
PDM_DIMMED_LED_LAMP_SetColorAdv(&PDM_DIMMED_LED_LAMP_0, 0x125); /* walk time = 3s */
```

   The PDM_DIMMED_LED_LAMP APP UI can be used to easily calculate the prescaler for the desired walk time (Figure 18).



PDM_DIMMED_LED_LAMP_0

Initial Fade Rates

Intensity linear walk time [ms]: 3000

Prescaler (LINPRES): 0x125

0-100% dimming transition time [ms]: 0.0

Prescaler (DLCK_PS): 0xDB

Divider (DIMDIV): 0x0

Figure 18 Use PDM_DIMMED_LED_LAMP APP to calculate linear walk time prescaler

3 Continuous Conduction Mode (CCM) buck LED drive

The CCM buck LED driver solution with the XMC1000 microcontroller uses an inverted buck topology. The inverted buck is a compact circuit consisting of a MOSFET, an inductor, a power diode and a shunt resistor. The high-current pads in the XMC1000 allow the microcontroller to drive the MOSFET directly. In cases where a higher current is required, an external gate driver can be placed between the XMC1000 and the MOSFET (Figure 19).

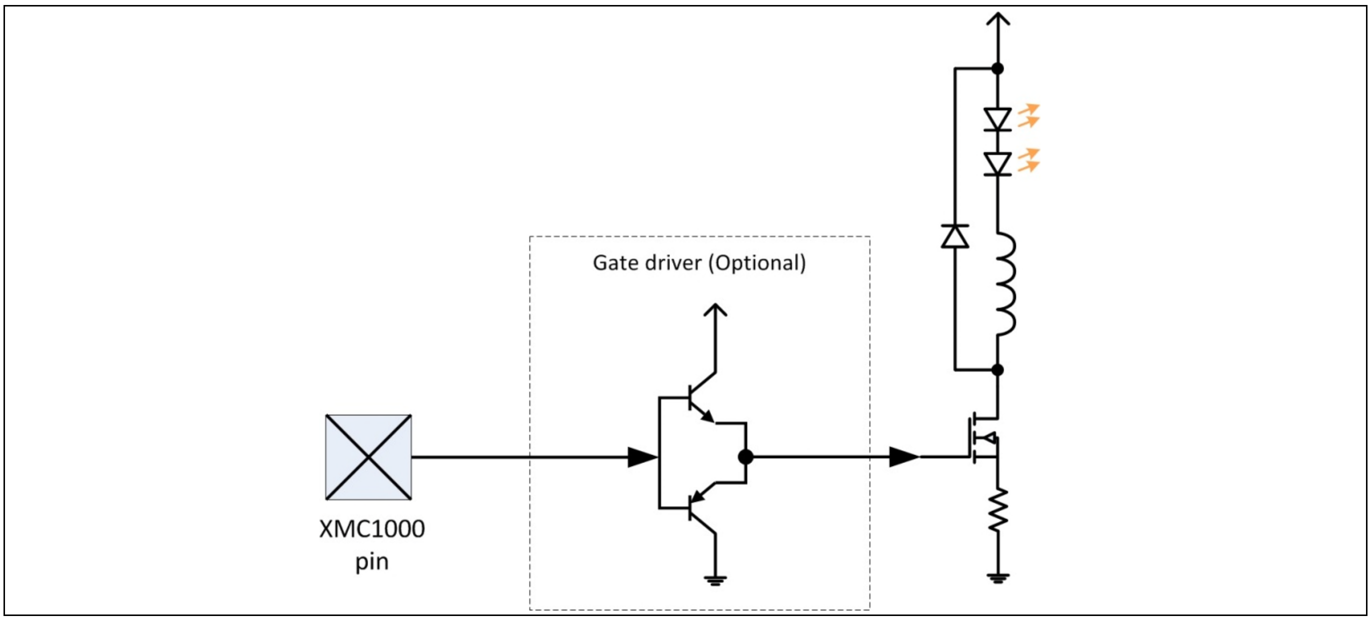


Figure 19 Inverted buck

The analog and switching peripherals in the XMC1000 microcontroller and the interconnections between them allow the implementation of a fast control loop to control the peak current of the LED channel. The control loop can be segmented to two parts – peak current control and modulation dimming (Figure 20).

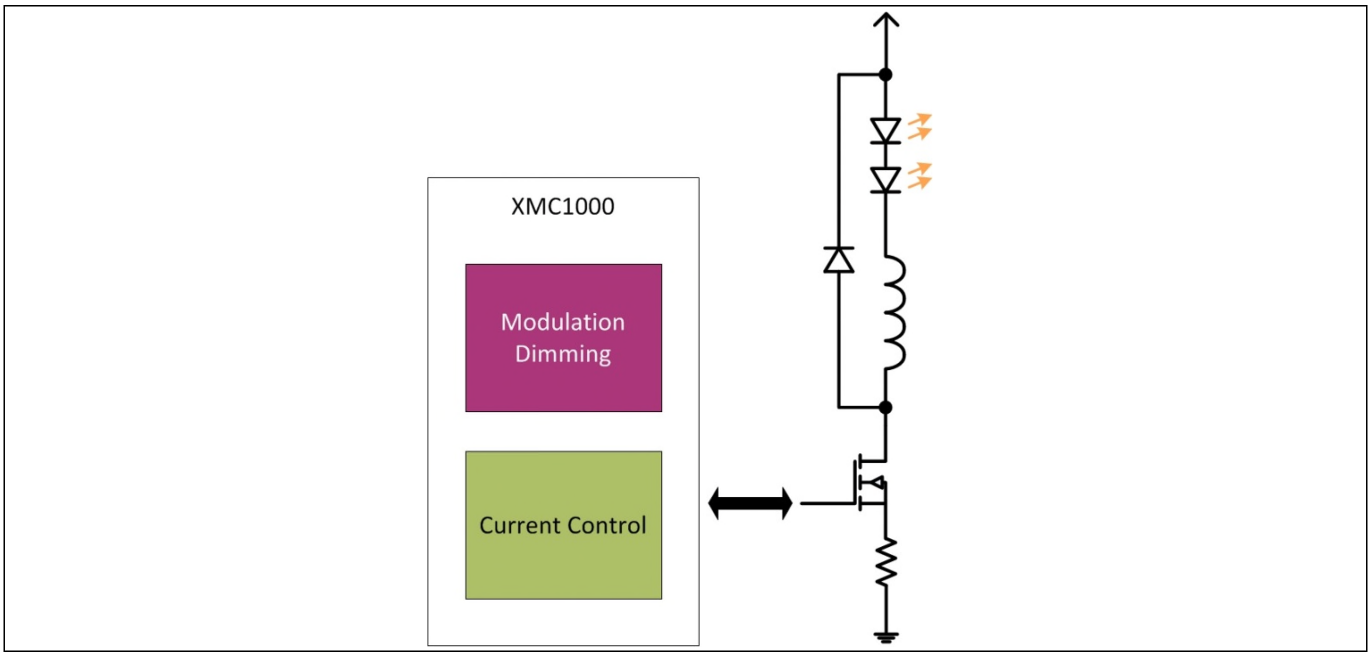


Figure 20 Control loop

3.1 Peak current control

The LED peak current is controlled using a combination of a Capture/Compare Unit 4 (CCU4) or a Capture/Compare Unit 8 (CCU8) slice and an Analog Comparator (ACMP) slice. The CCU4 or CCU8 timer slice generates the switching signal that drives the MOSFET gate (Figure 21). When the MOSFET is switched on, the current in the LED channel starts to rise. When the MOSFET is switched off, the current in the LED channel diminishes. Figure 22 provides an illustration of this relationship.

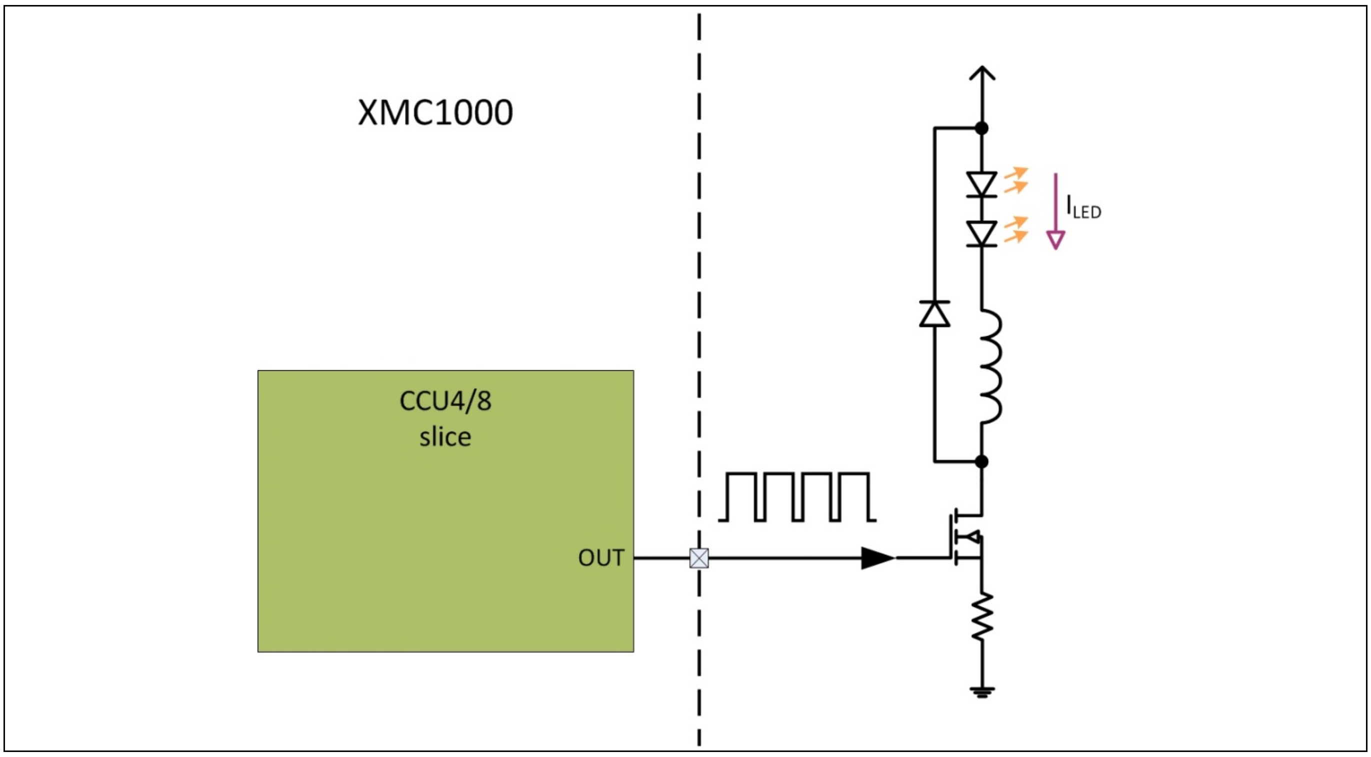


Figure 21 Switching output signal from CCU4 or CCU8 slice to drive MOSFET gate

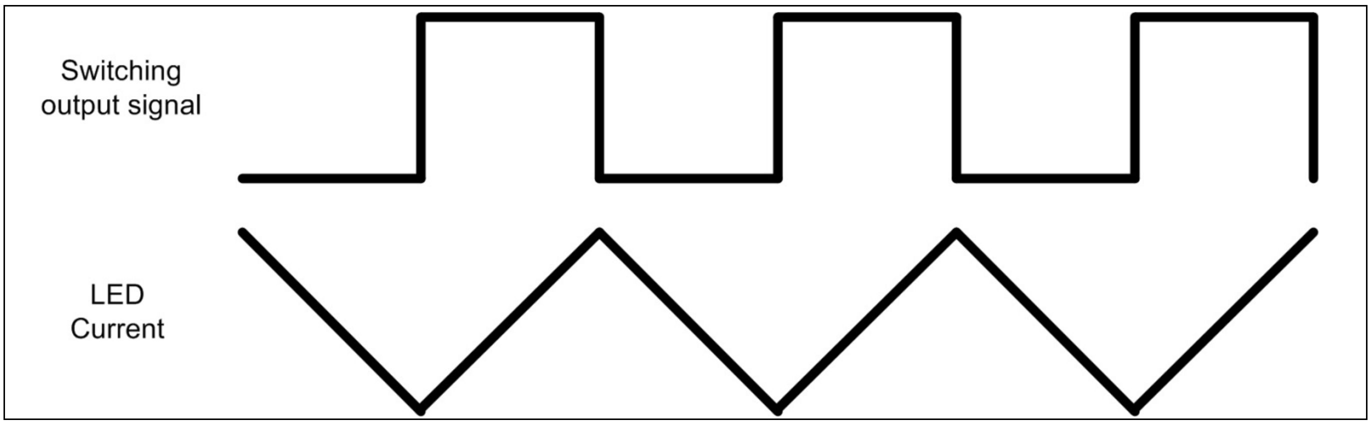


Figure 22 Switching signal from CCU4 or CCU8 slice and LED current

The ACMP slice acts as the LED peak current detector. The LED current is actively measured against a reference value that has been configured for the ACMP slice (Figure 23).

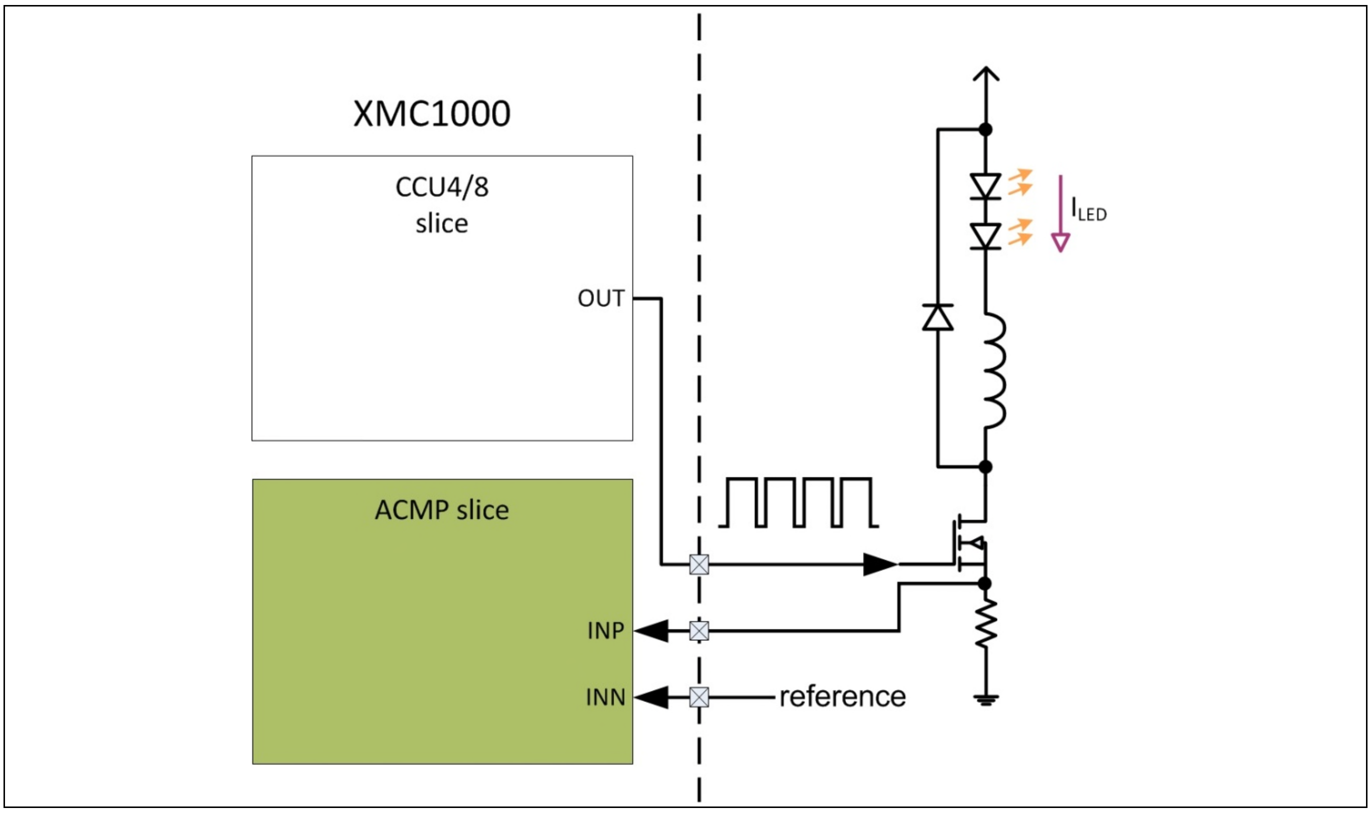


Figure 23 Peak current detection with ACMP slice

3.1.1 ACMP reference options

There are two options to configure the reference value to the ACMP slice.

3.1.1.1 External reference generation

The first option is via an external voltage divider circuit (Figure 24). The advantage of this option is that it is simple to implement. However, it would require re-soldering of new resistor values to change the reference value. The value generated may not be accurate due to the tolerance of the resistors. The value generated is also static, making it impossible to perform LED color control or dimming.

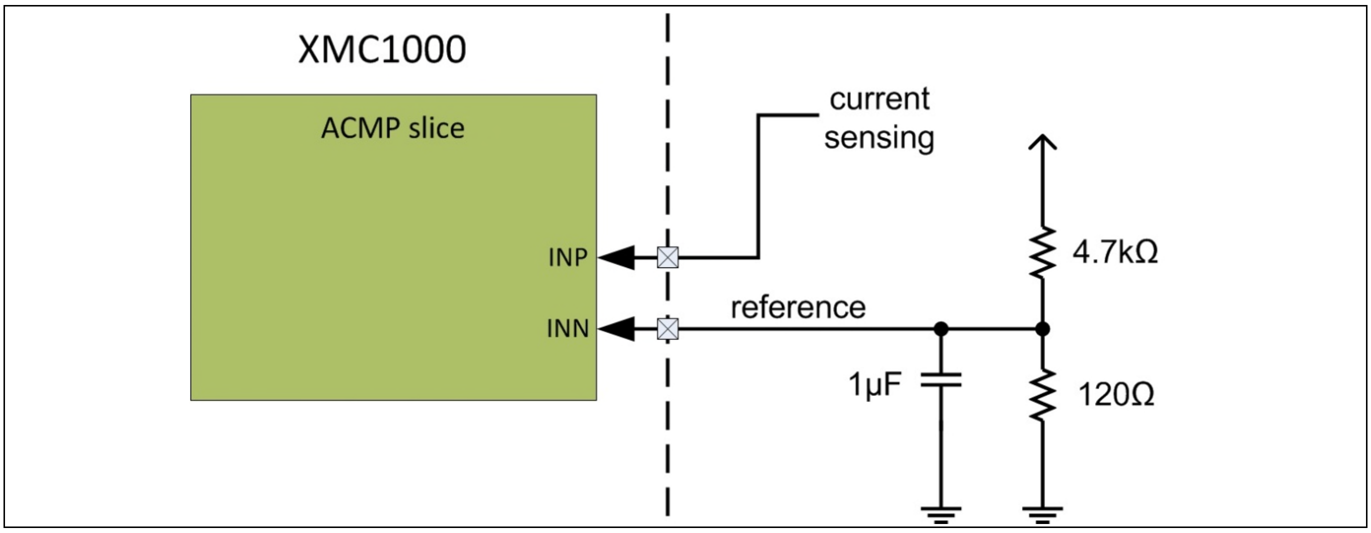


Figure 24 External reference value generation example for ACMP slice

3.1.1.2 Internal reference generation with external filter

The second option is via an internal reference generation with an external filter. This option involves a BCCU channel or a CCU4 (or CCU8) slice to generate a toggling output signal. The signal is then filtered with an external RC circuit to obtain an analog reference value for the ACMP slice (Figure 25), creating a pseudo Digital-to-Analog Converter (DAC). The advantage of this option is that the reference value can easily be changed via software programming to alter the frequency of the toggling signal. However, an additional microcontroller pin is required with this option.

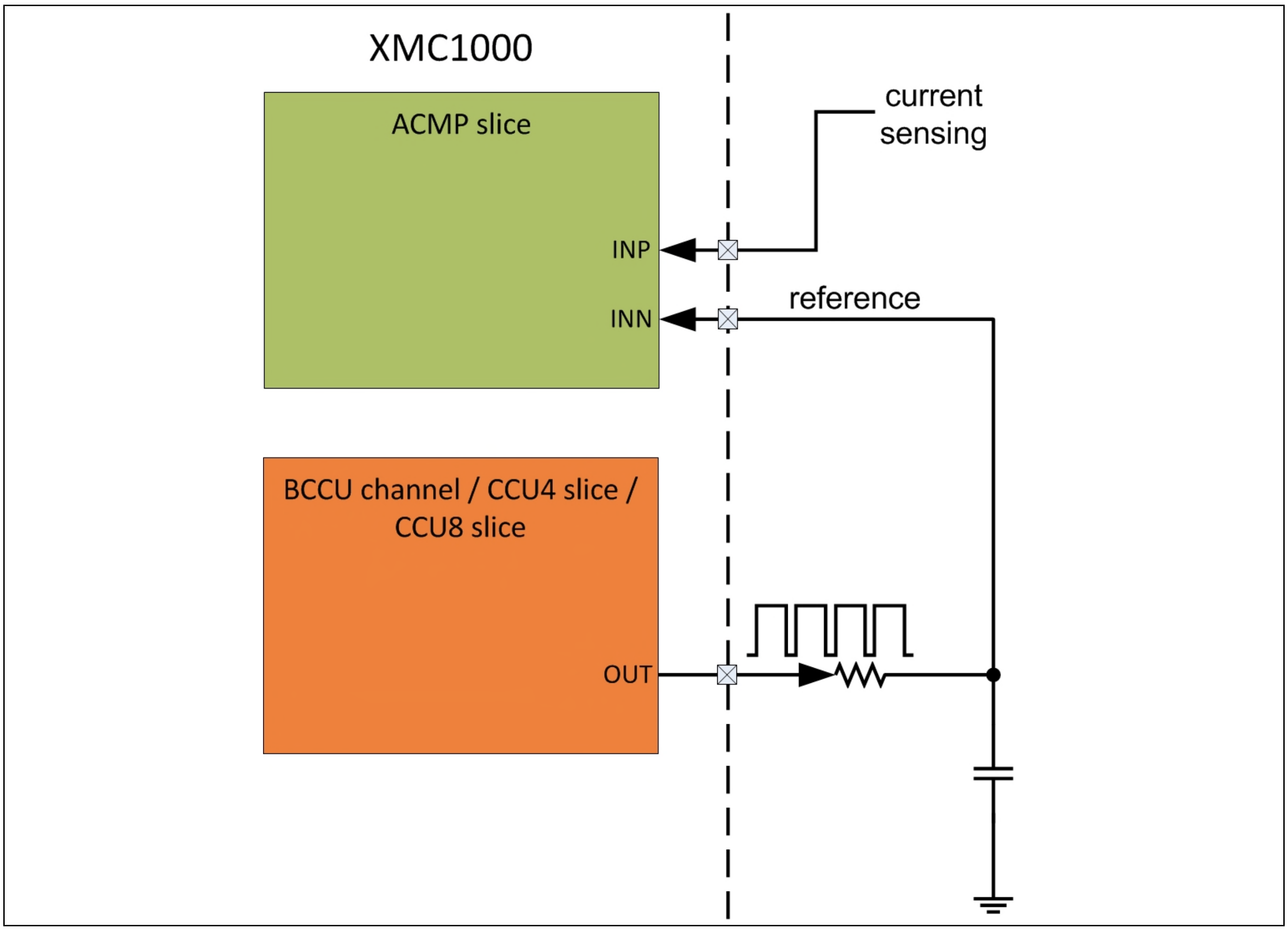


Figure 25 Internal reference with external RC filter

When the ACMP slice detects that the LED current has risen to the level of its reference, it toggles its output. This information can be used to trigger the CCUx slice to turn off its output so that the LED current will stop rising. On receiving this trigger from the ACMP, the CCUx slice will turn off its output for a fixed (pre-configured) time period before turning on again. This behavior can be realized in the CCUx using the external stop function with a “clear timer” configuration.

The LED current will drop during the CCUx off-time and start to rise again when the CCUx output returns active, resulting in a ripple in the LED current. The CCUx off-time will affect the ripple size. Figure 26 illustrates the interaction between the LED current, ACMP output and CCUx output.

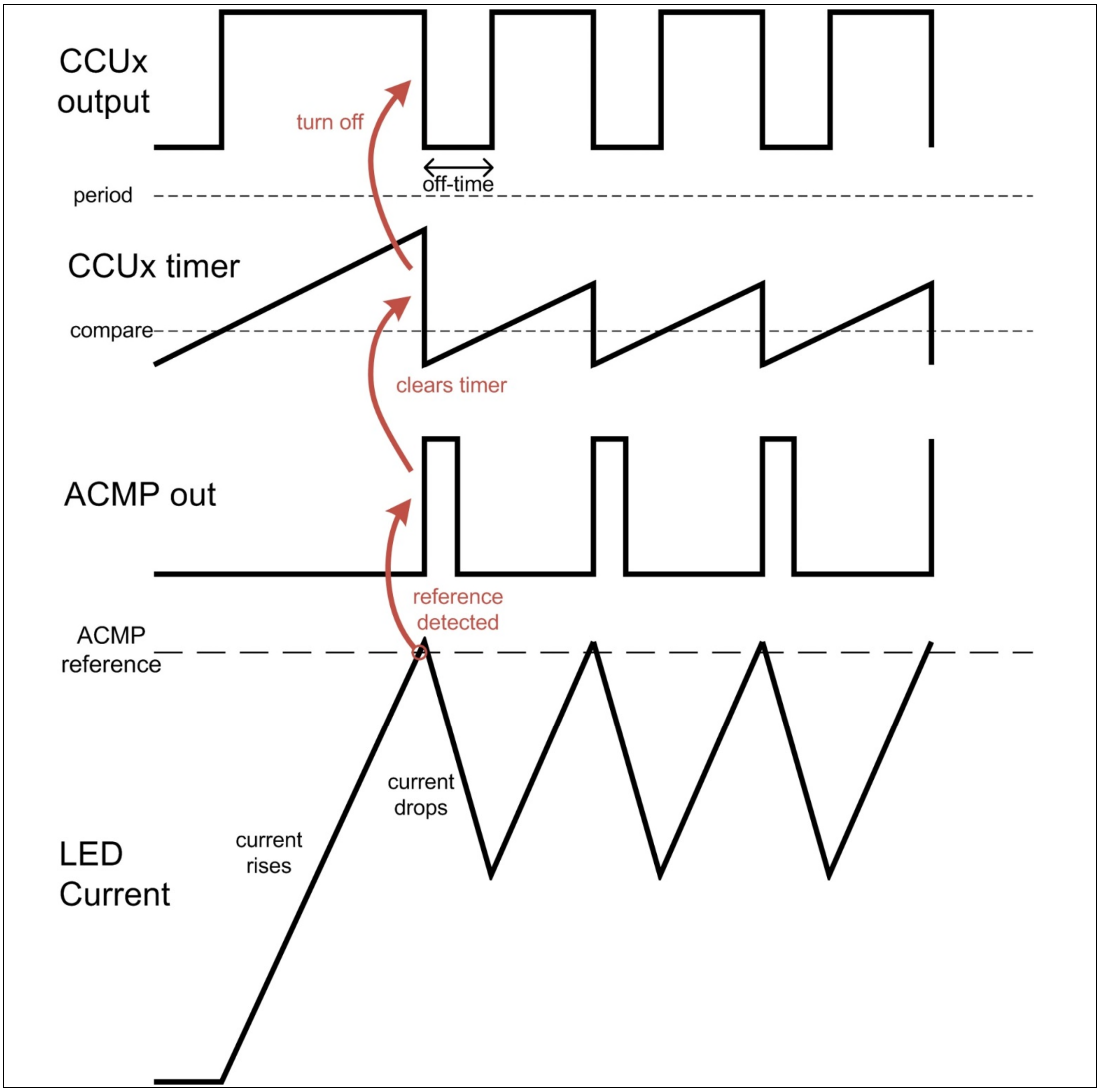


Figure 26 Peak current detection waveforms

3.1.2 ACMP output routing options

There are four possible routing options from the ACMP to the CCUx slice.

3.1.2.1 Option 1: via ERU slice

The first routing option involves using the Event Request Unit (ERU) to relay the ACMP output to the CCUx input (Figure 27).

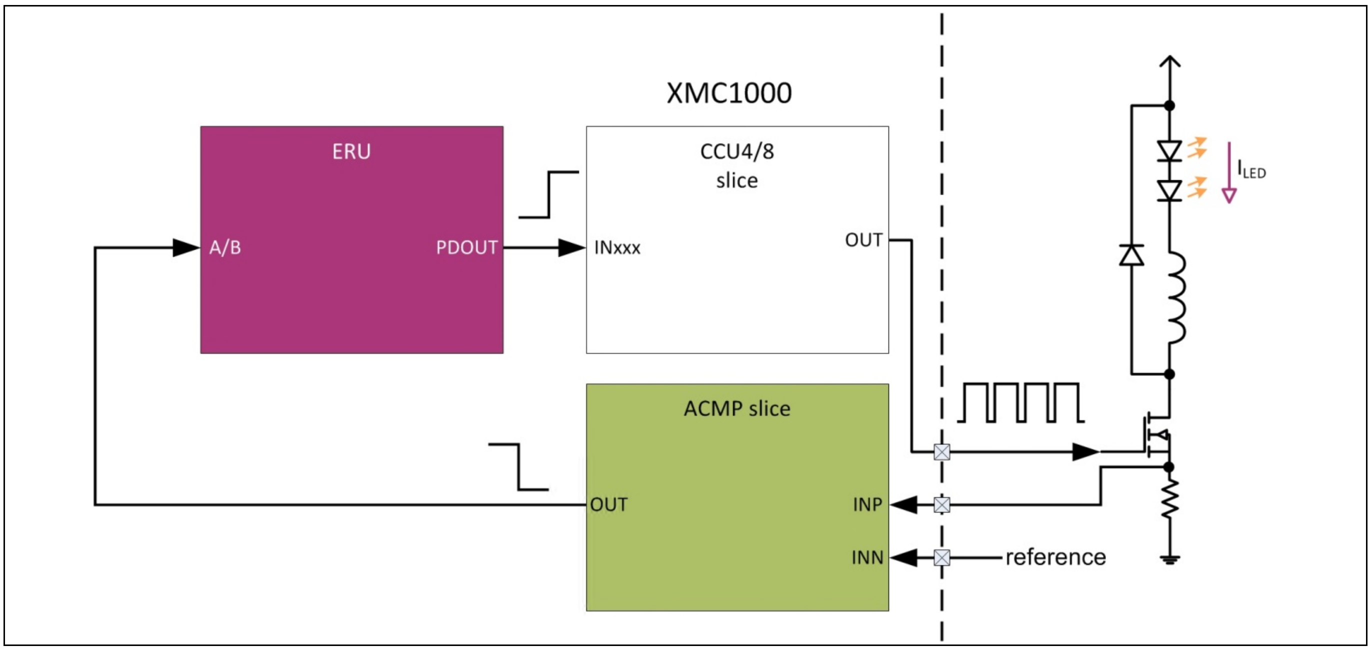


Figure 27 Option 1: routing ACMP to CCUx via ERU

3.1.2.2 Option 2: via physical pins

The second routing option involves mapping both the ACMP output and CCUx input to physical pins and shorting them together (Figure 28). This also implies that two microcontroller pins have to be reserved for this purpose.

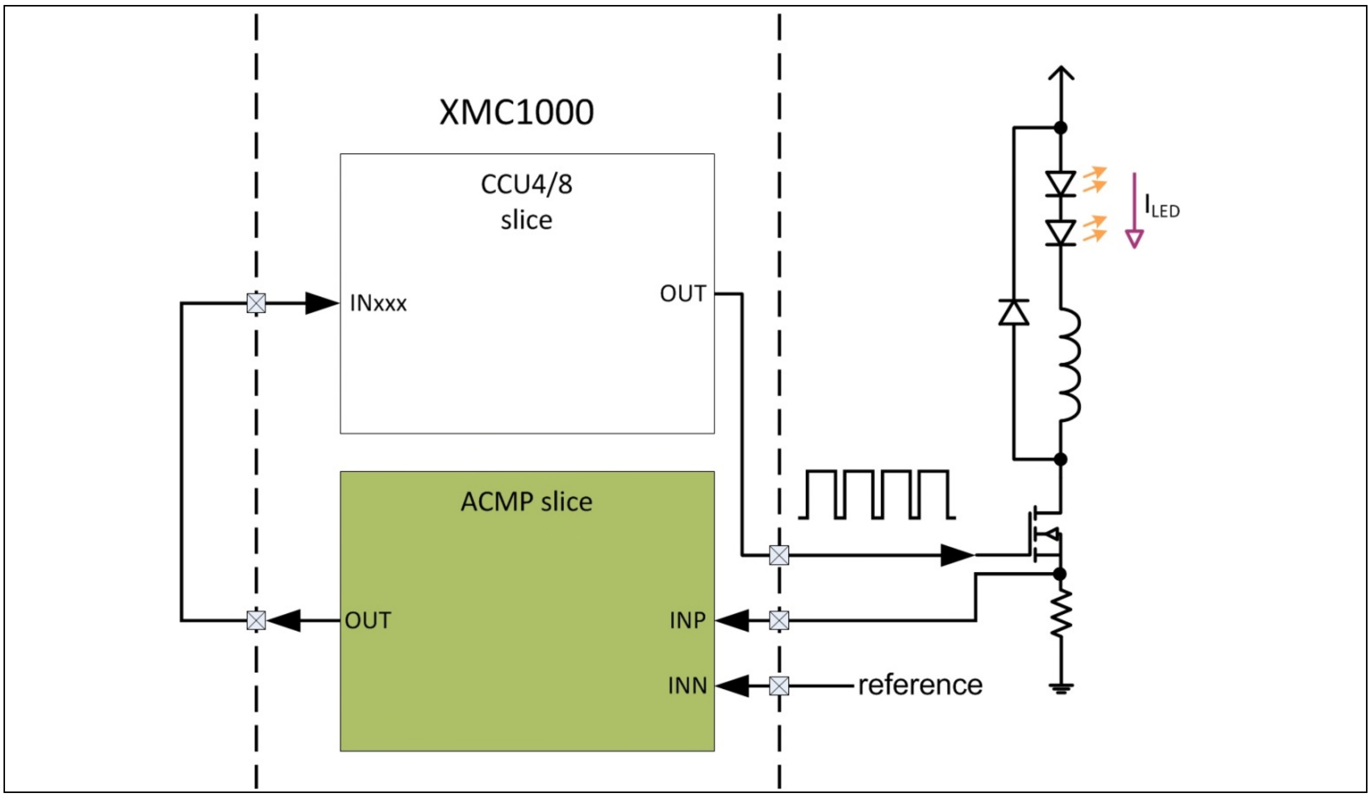


Figure 28 Option 2: routing ACMP to CCUx via physical pins

3.1.2.3 Option 4: via BCCU channel

The third routing option involves a BCCU channel to relay the ACMP output to the CCU8 slice (Figure 29). This option can only be implemented in the XMC130x and XMC140x devices.

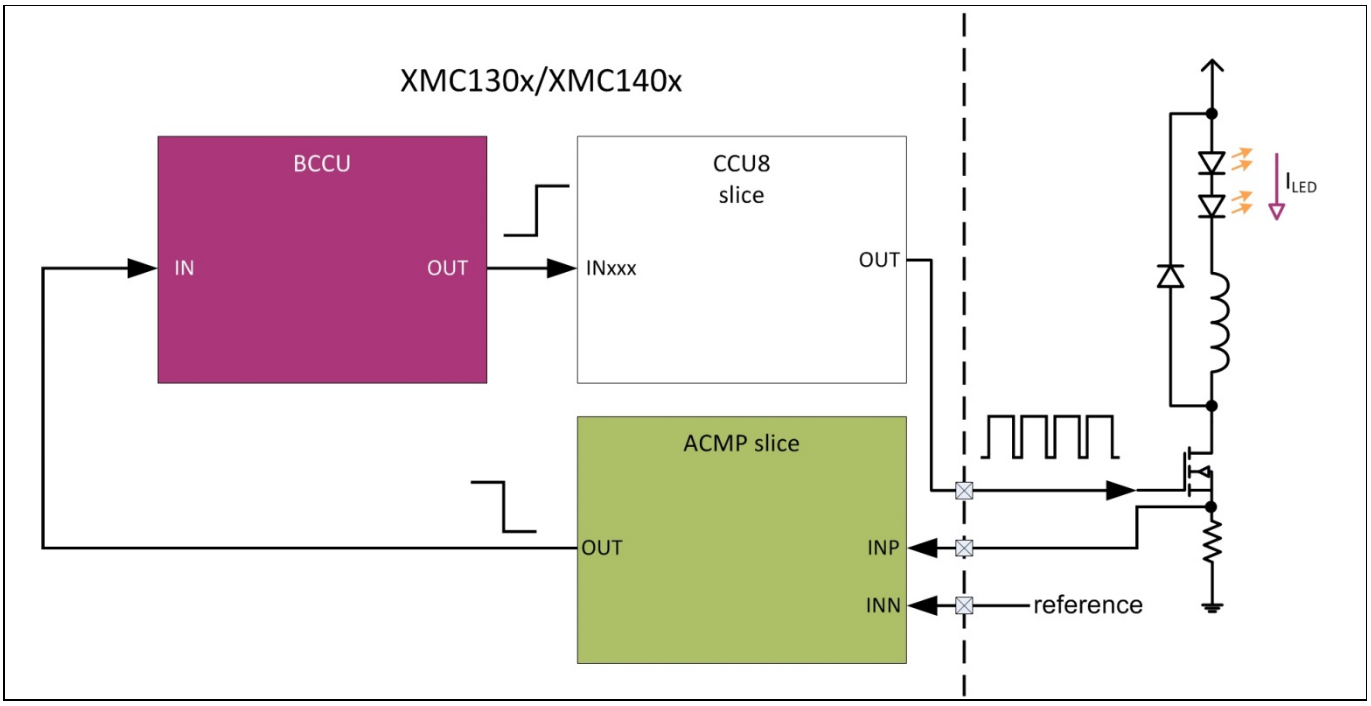


Figure 29 Option 3: routing ACMP to CCUx via BCCU channel

3.1.2.4 Option 4: direct connection

The final routing option involves connecting the ACMP output directly to the CCUx slice (Figure 30). This option is only available for XMC140x devices.

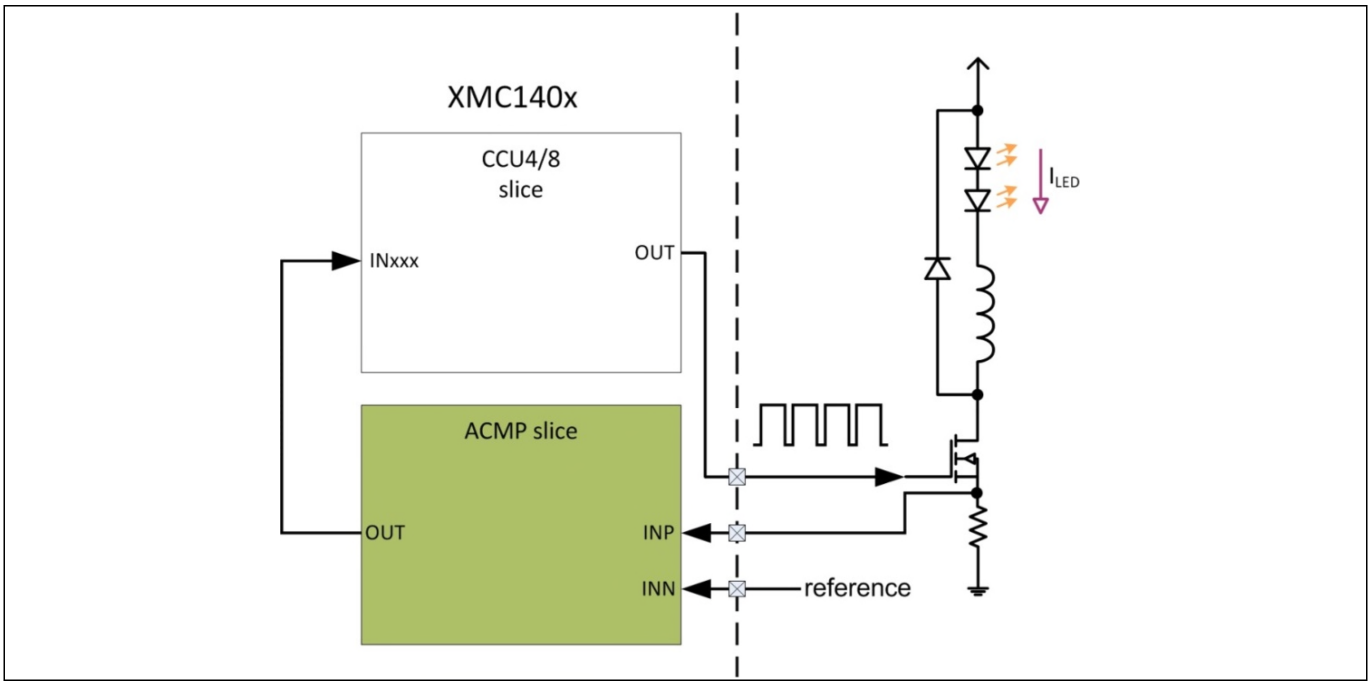


Figure 30 Option 4: direct routing of ACMP output to CCUx slice

3.1.3 Propagation delay

There is a propagation delay, which is defined as the measured time between the detection made by the ACMP slice to the turning off of the CCUx output. During this propagation delay, the LED current continues to rise, resulting in a current overshoot (Figure 31). The length of propagation delay, and thus the amount of current overshoot, varies across the four routing options. Table 2 provides an overview of the implications of the routing options on the propagation delay and current overshoot.

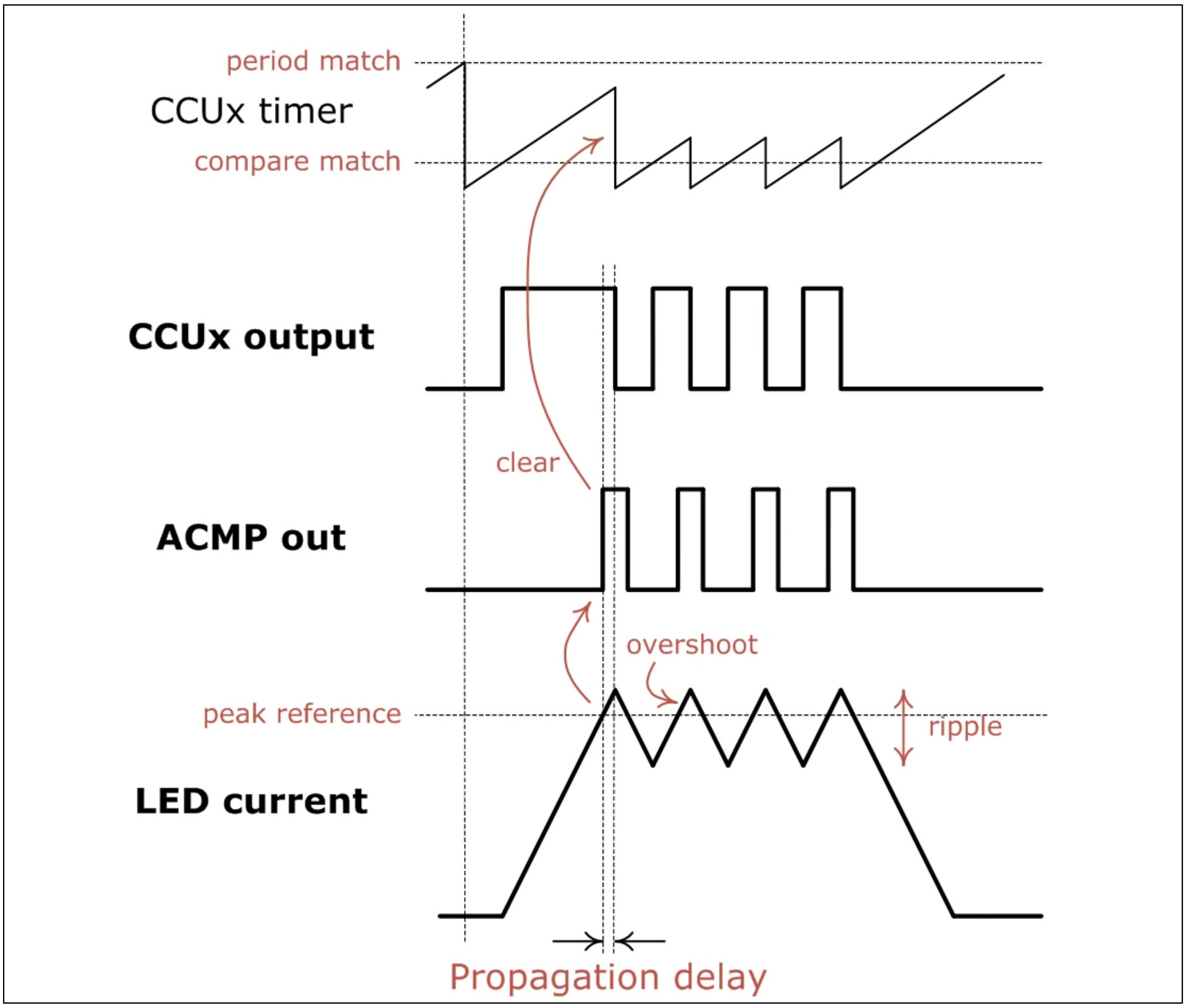


Figure 31 LED current overshoot

Table 2 Routing options and implications

Routing option	Propagation delay	Current overshoot	MOSFET switching speed	Availability in device
1	Longest	Largest	Slowest	XMC120x, XMC130x, XMC140x
2	Long	Large	Slow	XMC120x, XMC130x. XMC140x
3	Short	Short	Fast	XMC130x, XMC140x
4	Shortest	Shortest	Fastest	XMC140x

The magnitude of the current overshoot affects the size of the LED current ripple. It is essential that the LED current ripple is kept to the minimum to be able to switch the MOSFET at the highest speed, in order to achieve the best dimming performance (Figure 32). In Section 3.5 Tuning LED current ripple, a step-by-step guide for tuning the LED current ripple is provided.

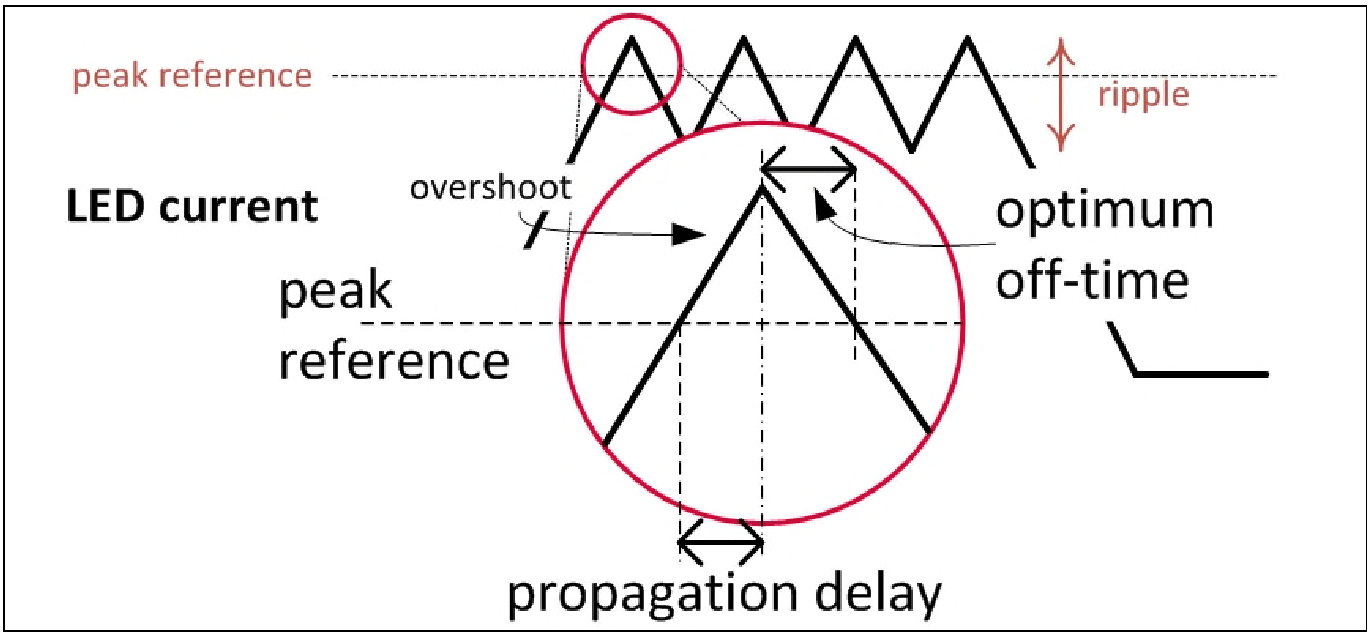


Figure 32 Optimum off-time

3.2 Modulation dimming

In this section, we will show how the modulation dimming feature is integrated into the peak current control loop.

The BCCU channel's PDM output contains the dimming level information. The PDM signal is used as a modulation input to the CCUx slice, meaning that the CCUx output is observed only during the PDM signal on-times (Figure 33).

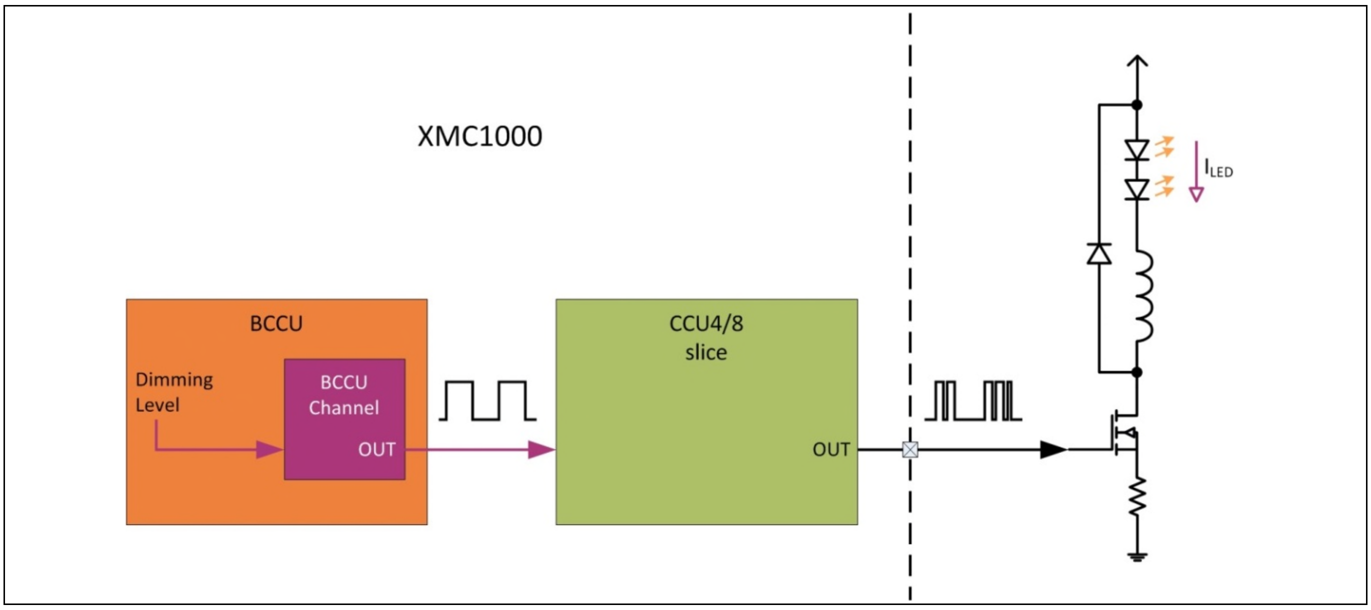


Figure 33 BCCU's PDM output as modulation to CCUx slice

The CCUx slice input that receives the PDM output varies across the XMC1000 devices due to the varying interconnections that are available across devices. For an XMC120x device, the PDM output is used as a multi-channel input (MCI) control signal to the CCU4 slice (Figure 34). For this, the multi-channel mode for the CCU4 slice has to be enabled. Figure 35 illustrates how the PDM signal interacts with the other signals in the peak current control loop.

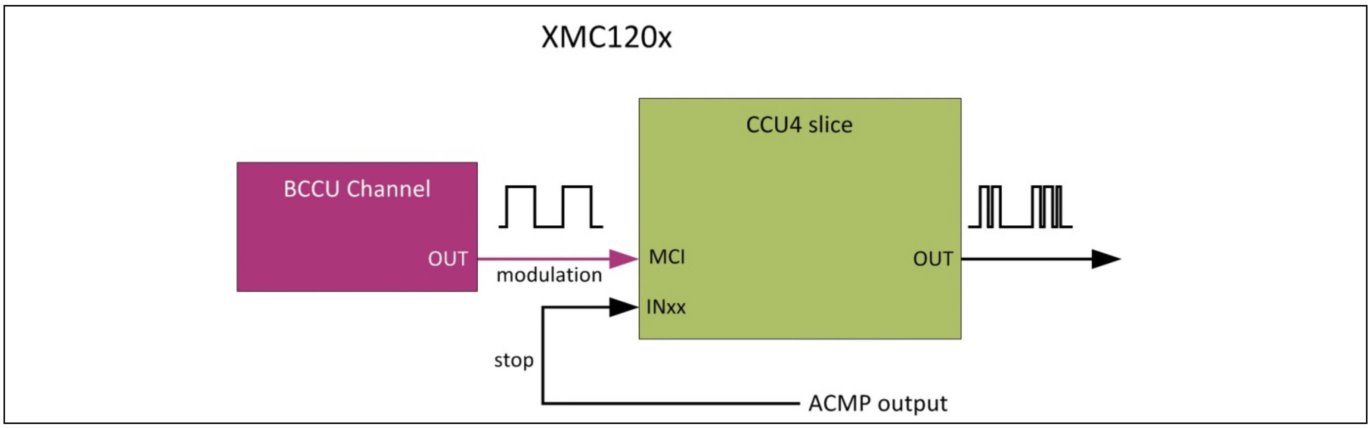


Figure 34 PDM output as MCI control signal for CCU4 slice in XMC120x

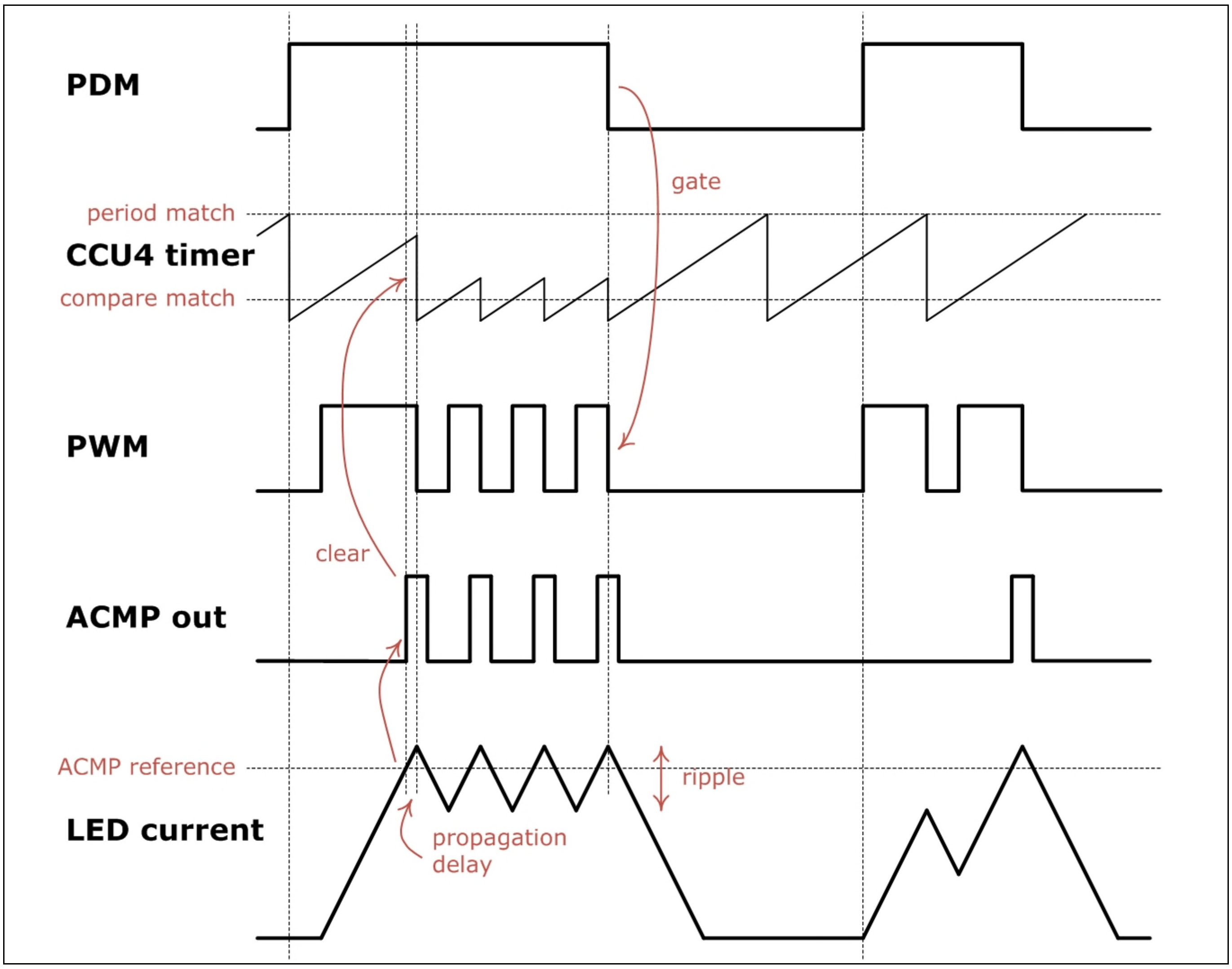


Figure 35 Modulation dimming with peak current control in XMC120x

Continuous Conduction Mode (CCM) buck LED drive

For an XMC130x or XMC140x device, the PDM output is used as an input to the CCU8 slice. As previously discussed in Section 3.1 Peak current control, in routing option 3, a BCCU channel is used to relay the ACMP output to the CCU8 slice. The same BCCU channel can be used to provide the modulation dimming signal to the CCU8 slice. To implement this, the ACMP output has to be inverted, and the PDM output is used as an input signal to the CCU8 slice for two external events – external stop and external modulation (Figure 36). Figure 37 illustrates how the PDM signal interacts with the other signals in the peak current control loop.

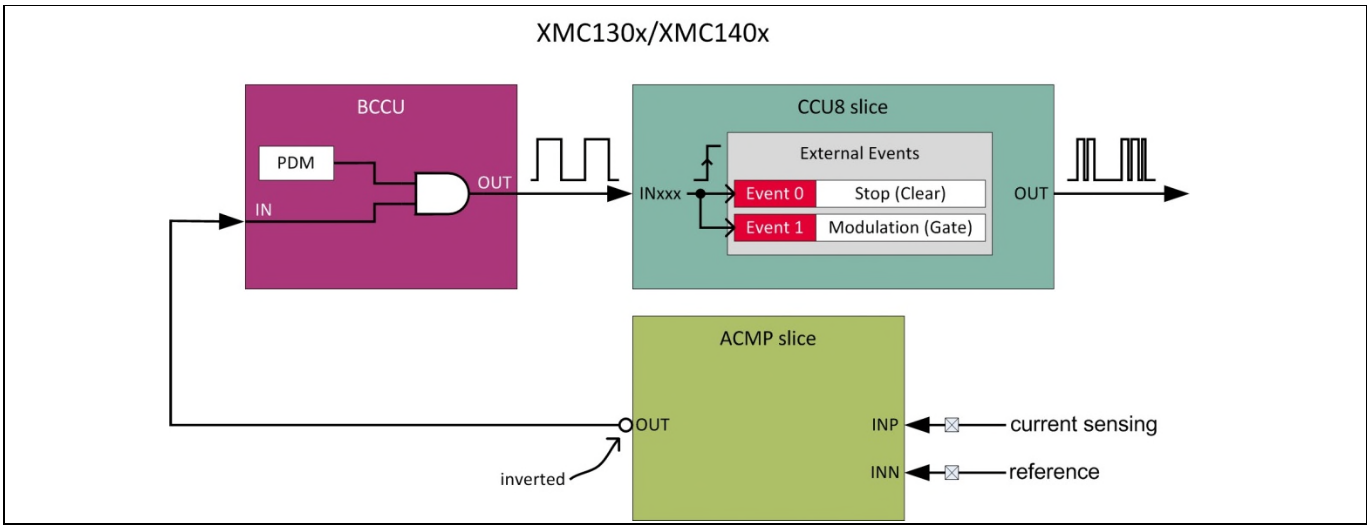


Figure 36 PDM output as input signal to CCU8 slice in XMC130x or XMC140x

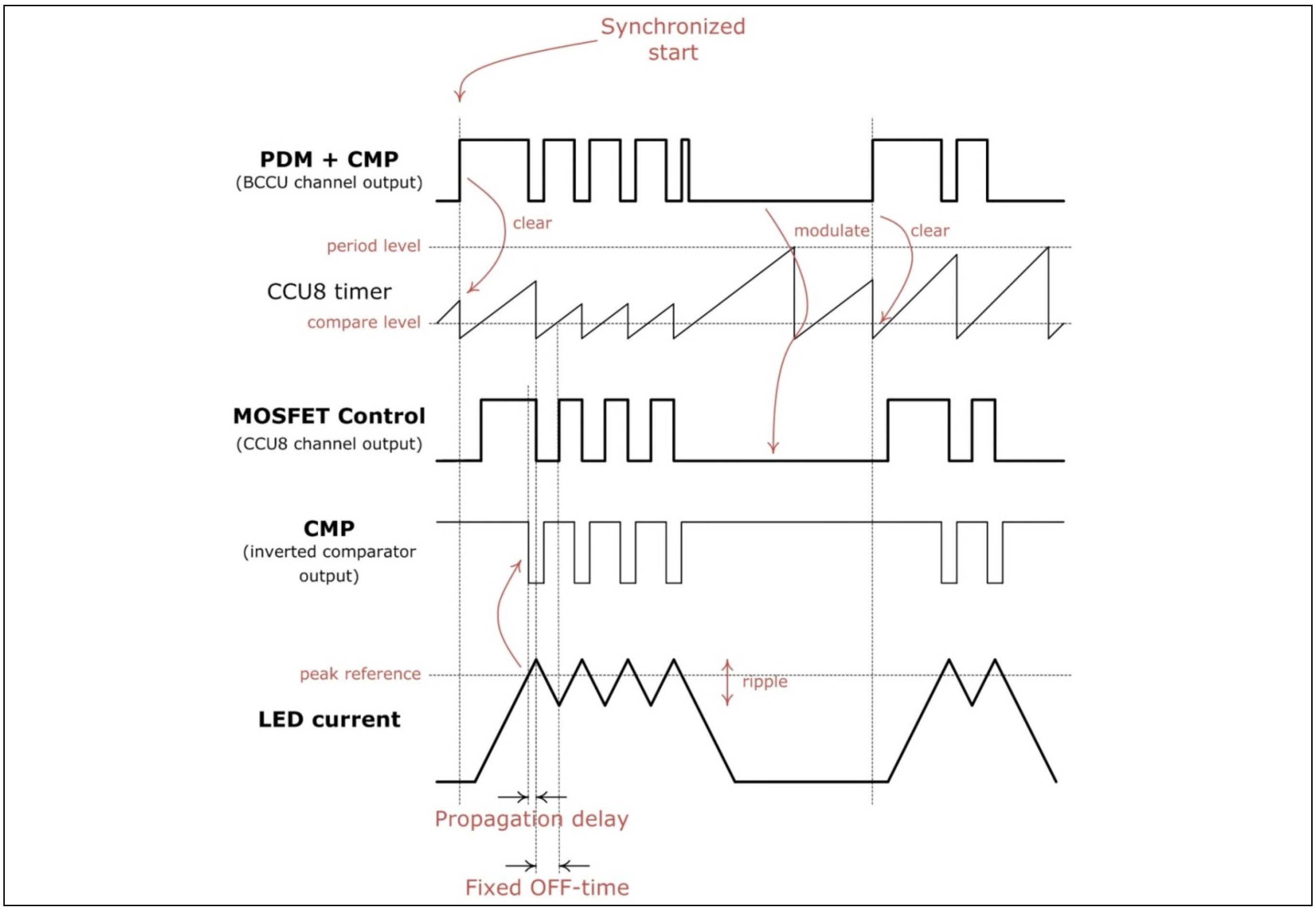


Figure 37 Modulation dimming with peak current control in XMC130x or XMC140x

3.3 Evaluation kits

Infineon currently offers two kits for evaluating the CCM buck LED Driver solution; the RGB LED lighting shield with XMC1202 for Arduino™ (Figure 38) and the XMC™ LED current control explorer (Figure 39).

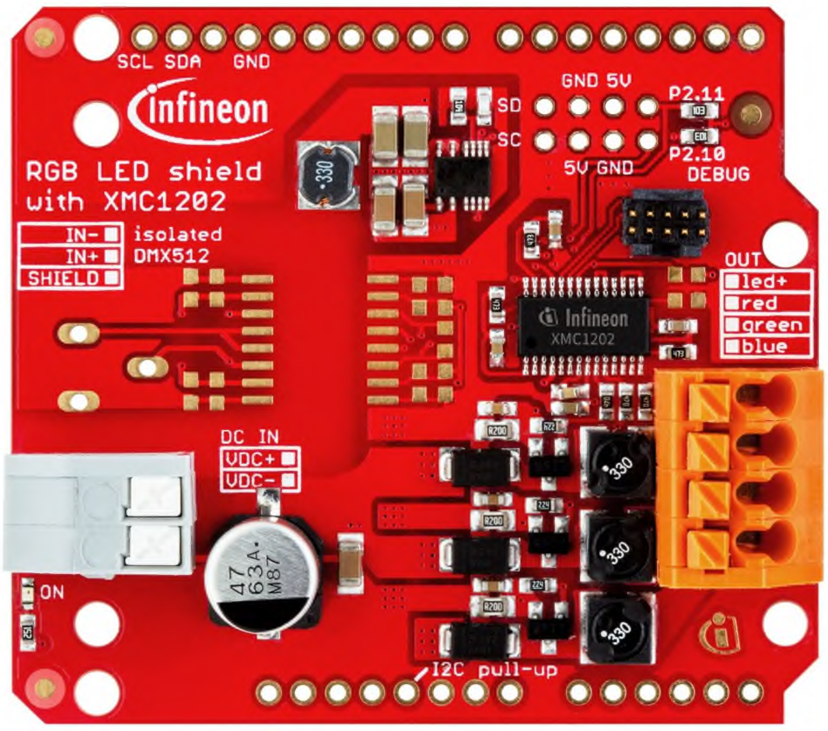


Figure 38 RGB LED lighting shield with XMC1202 for Arduino™

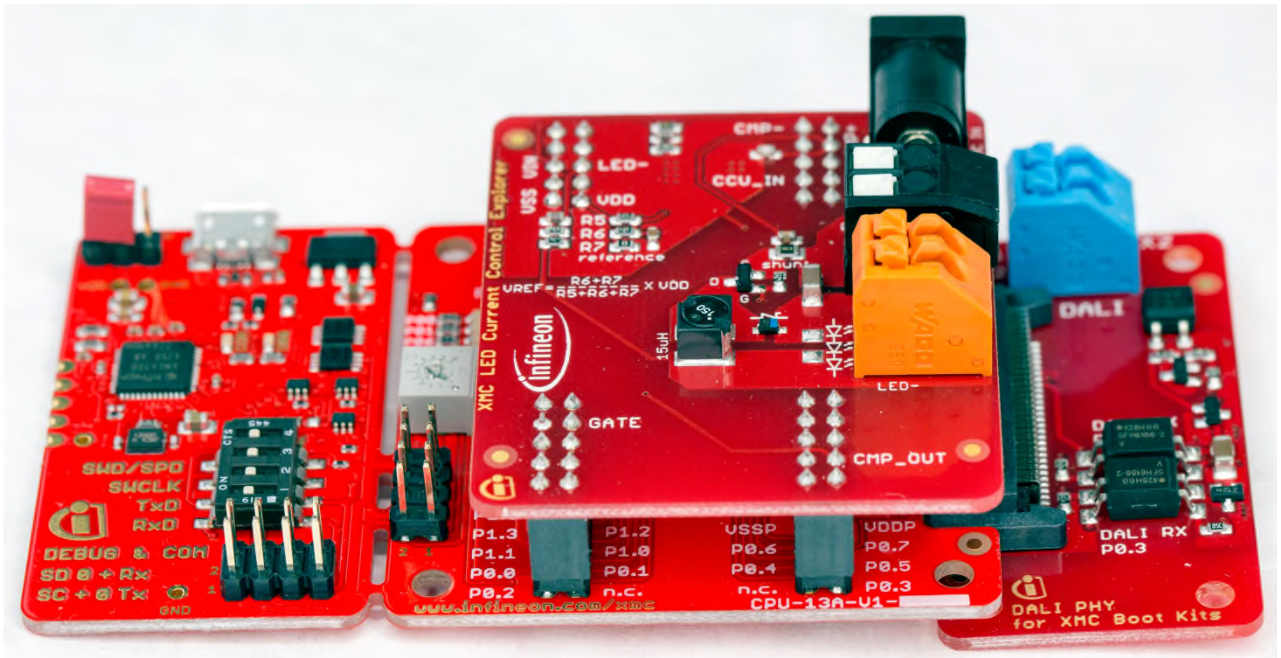


Figure 39 XMC™ LED current control explorer

Table 3 provides an overview of some of the features of these kits.

Table 3 Overview of features of evaluation kits

	RGB LED lighting shield with XMC1202 for Arduino™	XMC™ LED current control explorer
Microcontroller	XMC1202-T028X0016	XMC1302-T038X0200 AB
Other Infineon components	BSR606N	BSS306N BAS3010A-03W
No. of LED channels	3	1
Max. DC supply	48 V	30 V
Max Peak current	1 A per channel	1 A
Average current	700 mA per channel	700 mA
ACMP reference	Internal reference generation with external filter	External reference generation circuit
ACMP Output -> CCUx	Routing option 2	Routing option 1, 2 or 3

3.4 Example: driving an LED lamp using the XMC™ LED current control explorer

This section will demonstrate the steps required for software to drive a single-channel LED lamp using the XMC™ LED current control explorer.

The resources that are used in the XMC™ LED current control explorer are illustrated in Figure 40. These resources are selected or determined based on the available peripheral interconnections within the XMC1302 microcontroller. Refer to the interconnects section for each peripheral in the device's user manual for more information.

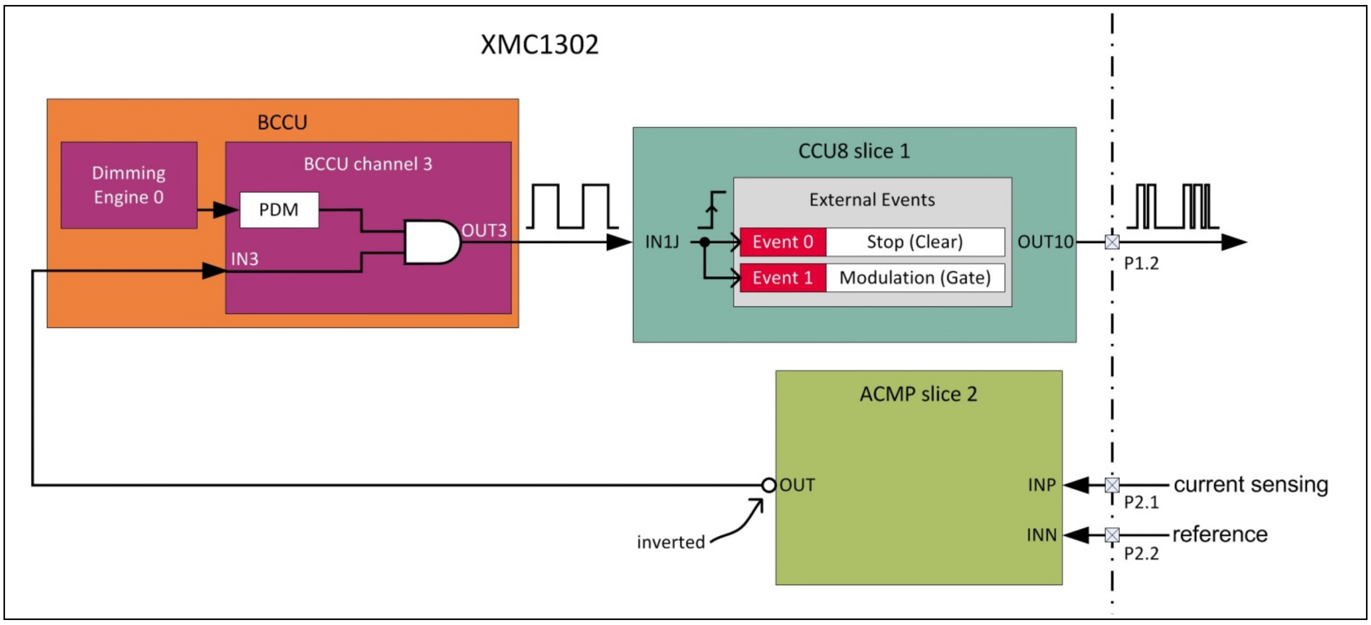


Figure 40 Resources used in XMC™ LED current control explorer

3.4.1 Implementation using XMC™ lib

3.4.1.1 ACMP reference generation

The XMC™ LED current control explorer does not require this step as it uses an external circuit for reference generation (as in Figure 24). As an example, if a BCCU channel is used as a pseudo Digital-to-Analog Converter (DAC) for ACMP reference generation (as in Figure 25):

File includes

```
#include <xmc_bccu.h>
```

Macro definitions

```
/* PDM output to be low pass filtered (DAC) for peak-current reference */
#define PDM_DAC P0_6
#define PDM_DAC_CHANNEL BCCU0_CH2
#define DAC_MASK 0x004 /* Channel 2 */
#define CURRENT_REF 70
```

Configuration data

```
/* Configuration for BCCU channel used as DAC (digital-analog convertor) for ACMP reference
generation */
XMC_BCCU_CH_CONFIG_t dac_config =
{
  .pack_en     = 0, /**< packer is disabled */
  .dim_bypass  = 1, /**< dimming is bypassed */
  .flick_wd_en = 0x0 /**< flicker watchdog is disabled */
};
```

Initialization

```
/* BCCU global initialization */
XMC_BCCU_GlobalInit(BCCU0, &bccu_global_config);

/* init BCCU channel to function as DAC */
XMC_BCCU_CH_Init(PDM_DAC_CHANNEL, &dac_config);

/* Set immediate linear walk for DAC */
XMC_BCCU_CH_SetLinearWalkPrescaler(PDM_DAC_CHANNEL, 0);
/* Enable DAC channel */
XMC_BCCU_ConcurrentEnableChannels(BCCU0, DAC_MASK);

/* set current reference to low now */
XMC_BCCU_CH_SetTargetIntensity(PDM_DAC_CHANNEL, CURRENT_REF);
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, DAC_MASK);
XMC_GPIO_SetMode(PDM_DAC, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT1);
```

More information on setting up a BCCU channel or a CCUx slice as a pseudo DAC is available in AN_201511_PL30_0011.

3.4.1.2 ACMP as peak current detector

The XMC™ LED current control explorer uses ACMP slice 2. The ACMP output is routed to the CCU4 slice input via option 3 (as in Figure 29), therefore the output is inverted. The implementation steps are as follows:

File includes

```
#include <xmc_acmp.h>
```

Configuration data

```
/* ACMP slice configuration */
XMC_ACMP_CONFIG_t cmp_config =
{
  .filter_disable = 0U, /**< Enable filter */
  .output_invert  = 1U, /**< Invert output */
  .hysteresis     = XMC_ACMP_HYSTERESIS_10 /**< Use 10mV hysteresis */
};
```

Initialization

```
/* Comparator initialization */
XMC_ACMP_Init(XMC_ACMP0, 2, &cmp_config);
XMC_ACMP_EnableComparator(XMC_ACMP0, 2);
XMC_ACMP_SetInput(XMC_ACMP0, 2, XMC_ACMP_INP_SOURCE_STANDARD_PORT);
```

3.4.1.3 BCCU for modulation dimming and ACMP route

The BCCU channel takes in the ACMP output as its gating input signal for route option 3 (see Figure 36). Therefore, the gating function has to be enabled for the BCCU channel. BCCU channel 3 is used as ACMP slice 2’s output is connected to this channel.

File includes

```
#include <xmc_bccu.h>
```

Configuration data

```
/* BCCU global configuration */
XMC_BCCU_GLOBAL_CONFIG_t bccu_global_config =
{
  .fclk_ps  = 0x13, /**< 3.333MHz @PCLK=64MHz */
  .dclk_ps  = 0xdb, /**< 292.237KHz @PCLK=64MHz */
  .bclk_sel = XMC_BCCU_BCLK_MODE_NORMAL, /**< 250KHz @PCLK=64MHz */
  .maxzero_at_output = 1000 /**< 0.1% min brightness */
};

/* BCCU trigger configuration */
XMC_BCCU_TRIG_CONFIG_t bccu_trig_config =
{
  .mode = XMC_BCCU_TRIGMODE0, /* trigger mode 0 */
  .delay = XMC_BCCU_TRIGDELAY_NO_DELAY, /**< trigger without delay */
  .mask_chans = 0x08 /**< channel 3 */
};

/* Configuration for BCCU channel used as PDM (pulse-density modulator) */
XMC_BCCU_CH_CONFIG_t pdm_config =
{
  .pack_en          = 0, /**< disable packer */
  .force_trig_en    = 1, /**< enable forced trigger */
  .trig_edge        = XMC_BCCU_CH_TRIG_EDGE_PASS_TO_ACT, /**< trigger on rising edge */
  .dim_sel          = 0x0, /**< Use dimming engine 0 */
  .dim_bypass       = 0, /**< disable bypass */
  .gate_en          = 1, /** enable gating */
  .flick_wd_en      = 1, /**< enable flicker watchdog */
  .trig_edge        = XMC_BCCU_CH_TRIG_EDGE_PASS_TO_ACT, /**< rising edge */
  .pack_offcnt_val = 0, /**< initial packer off count value */
  .pack_offcmp_lev = 199, /**< initial packer off compare value */
  .pack_oncmp_lev  = 3 /**< initial packer on compare value */
};

/* Configuration for BCCU dimming engine */
XMC_BCCU_DIM_CONFIG_t bccu_dim_config =
{
  .dim_div = 0x0 /**< dimming engine divider value = 0s */
};
```

Initialization

```
/* BCCU global initialization */
XMC_BCCU_GlobalInit(BCCU0, &bccu_global_config);

/* init dimming engine */
XMC_BCCU_DIM_Init(BCCU0_DE0, &bccu_dim_config);

/* enable dimming engine */
XMC_BCCU_EnableDimmingEngine(BCCU0, XMC_BCCU_CH_DIMMING_SOURCE_DE0);

/* init PDM channel */
XMC_BCCU_CH_Init(BCCU0_CH3, &pdm_config);

/* Set fast (immediate) linear walk for PDM */
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH3, 0x00);

/* configure trigger for PDM channel */
XMC_BCCU_ConcurrentConfigTrigger(BCCU0, &bccu_trig_config);

/* Enable PDM channel */
XMC_BCCU_ConcurrentEnableChannels(BCCU0, 0x08);

/* set initial intensity to 100% */
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH3, 4095);

/* start linear walk to effect intensity change */
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x08);
```

3.4.1.4 CCU8 as MOSFET gate driver

This section provides the software to configure the CCU8 slice's external events for the realization of the modulation dimming and current control features.

File includes

```
#include <xmc_gpio.h>
#include <xmc1_gpio.h>
#include <xmc_ccu8.h>
```

Macro definitions

```
#define OFFTIME (0x7) /* MOSFET off-time = 110ns */
#define GATE P1_2 /* CCU4 output pin (connected to MOSFET gate) */
```

Configuration data

```
/* CCU8 configuration for MOSFET gate driver */
XMC_CCU8_SLICE_COMPARE_CONFIG_t ccu8_mosfet_gate_config =
{
  .timer_mode          = XMC_CCU8_SLICE_TIMER_COUNT_MODE_EA, /**< edge-aligned mode */
  .monoshot            = 0, /**< no single shot */
  .shadow_xfer_clear   = 0U, /**< no shadow transfer when timer clears */
  .dither_timer_period = 0U, /**< no dither for timer period */
  .dither_duty_cycle   = 0U, /**< no dither for timer compare */
  .prescaler_mode      = XMC_CCU8_SLICE_PRESCALER_MODE_NORMAL, /**< normal prescaler mode */
  .mcm_ch1_enable      = 0U, /**< disable multi-channel mode */
  .mcm_ch2_enable      = 0U, /**< disable multi-channel mode */
  .slice_status          = XMC_CCU8_SLICE_STATUS_CHANNEL_1, /**< Channel 1 to drive slice
status output */
  .passive_level_out1  = XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW, /**< low passive level */
  .asymmetric_pwm      = 0U, /**< PWM to be a function of period value */
  .invert_out1         = 0U, /**< Do not use inverted ST of Channel 1 */
  .prescaler_initval   = 0U, /**< initial prescaler value */
  .float_limit         = 0U, /**< not used */
  .dither_limit        = 0U, /**< not used */
  .timer_concatenation = 0U /**< not used */
};

/* Configuration for CCU8 slice external event 0 */
XMC_CCU8_SLICE_EVENT_CONFIG_t slice_clear_event_config =
{
  .mapped_input     = CCU80_IN1_BCCU0_OUT3, /**< input signal that will trigger this event */
  .edge             = XMC_CCU8_SLICE_EVENT_EDGE_SENSITIVITY_RISING_EDGE, /**< trigger on
rising edge */
  .duration  = XMC_CCU8_SLICE_EVENT_FILTER_DISABLED /**< disable filter */
};

/* Configuration for CCU8 slice external event 1 */
XMC_CCU8_SLICE_EVENT_CONFIG_t slice_mod_event_config =
{
  .mapped_input     = CCU80_IN1_BCCU0_OUT3, /**< input signal that will trigger this event */
  .level     = XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW, /**< active on low */
  .duration  = XMC_CCU8_SLICE_EVENT_FILTER_DISABLED /**< disable filter */
};
```

Initialization

```
/* Initialization for MOSFET gate driver */
/* Ensure that fCCU reaches CCU80 */
XMC_CCU8_SetModuleClock(CCU80, XMC_CCU8_CLOCK_SCU);

/* Initialize CCU80 kernel */
XMC_CCU8_Init(CCU80, XMC_CCU8_SLICE_MCMS_ACTION_TRANSFER_PR_CR);

/* Get the slice out of idle mode */
XMC_CCU8_EnableClock(CCU80, 1);

/* Start the prescaler and restore clock to slice */
XMC_CCU8_StartPrescaler(CCU80);

/* Initialize the slice */
XMC_CCU8_SLICE_CompareInit(CCU80_CC81, &ccu8_mosfet_gate_config);

/* Program period value */
XMC_CCU8_SLICE_SetTimerPeriodMatch(CCU80_CC81, 0x27FU);

/* Program compare value */
XMC_CCU8_SLICE_SetTimerCompareMatch(CCU80_CC81, XMC_CCU8_SLICE_COMPARE_CHANNEL_1, OFFTIME);
```

```
/* Enable shadow transfer */
XMC_CCU8_EnableShadowTransfer(CCU80, XMC_CCU8_SHADOW_TRANSFER_SLICE_1);

/* init external event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC81, XMC_CCU8_SLICE_EVENT_0,&slice_clear_event_config);

/* init stop event */
XMC_CCU8_SLICE_StopConfig(CCU80_CC81, XMC_CCU8_SLICE_EVENT_0,
XMC_CCU8_SLICE_END_MODE_TIMER_CLEAR);

/* init external event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC81, XMC_CCU8_SLICE_EVENT_1, &slice_mod_event_config);

/* init modulation event */
XMC_CCU8_SLICE_ModulationConfig(CCU80_CC81, XMC_CCU8_SLICE_EVENT_1,
XMC_CCU8_SLICE_MODULATION_MODE_CLEAR_OUT, XMC_CCU8_SLICE_MODULATION_CHANNEL_1, 0);

/* start switching */
XMC_CCU8_SLICE_StartTimer(CCU80_CC81);

/* enable output pin */
XMC_GPIO_SetMode(GATE, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5);
```

3.4.1.5 Application code

This section provides a guide to adjust the dimming level and color of the LED lamp, the ACMP reference value and the CCU8 off-time in-application.

1. Dimming Level

   To change the lamp brightness, the target dimming level must be set first:

   ```
   /* set target dimming level to 50% brightness */
   XMC_BCCU_DIM_SetTargetDimmingLevel(BCCU0_DE0, 2048);
   ```

   After the target dimming level is set, the change in dimming level can be effected by starting the dimming process:

   ```
   /* start dimming process */
   XMC_BCCU_StartDimming(BCCU0, XMC_BCCU_CH_DIMMING_SOURCE_DE0);
   ```

   To change the fade time, use the following function before starting the dimming process:

   ```
   /* set fade time to 3s */
   XMC_BCCU_DIM_SetDimDivider(BCCU0_DE0, 0x2B);
   ```

2. Color (applies only in the case of a multi-channel color LED lamp)

   To change the color of the LED, first set the target channel intensities. The combination of the channel intensities results in a color. One simple color scheme that can be suggested here is as follows (Table 4), whereby the respective channel intensities sum to a maximum intensity value of 4095 or 4096.

Table 4 Simple color scheme

Desired color	RED channel intensity	GREEN channel intensity	BLUE channel intensity
Red	4095	0	0
Green	0	4095	0
Blue	0	0	4095
White	1365	1365	1365

Desired color	RED channel intensity	GREEN channel intensity	BLUE channel intensity
Yellow	2048	2048	0
Magenta	2048	0	2048
Cyan	0	2048	2048

For example, to change the color of the LED to red:

```
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 4095);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH7, 0);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH8, 0);
```

Start the linear walk concurrently in all channels.

```
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x181);
```

3. ACMP reference value (applies only if internal reference generation is used)
   If a BCCU channel is used, the channel intensity required to generate a particular reference voltage can be calculated:

$$Intensity = \frac{V_{Reference} \times 4095}{V_{Operating}}$$

As an example, to set the calculated intensity:
```
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 82);
```

To effect the change:
```
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x001);
```

If a CCU4 slice is used, the timer compare value required to generate a particular reference voltage can be calculated:

$$Compare = \left(1 - \left(\frac{V_{Reference}}{V_{Operating}}\right)\right) \times (PeriodVal + 1)$$

As an example, to set the calculated compare value:
```
XMC_CCU4_SLICE_SetTimerCompareMatch(CCU40_CC42, 601);
```

To effect the change:
```
XMC_CCU4_EnableShadowTransfer(CCU40, XMC_CCU4_SHADOW_TRANSFER_SLICE_2);
```

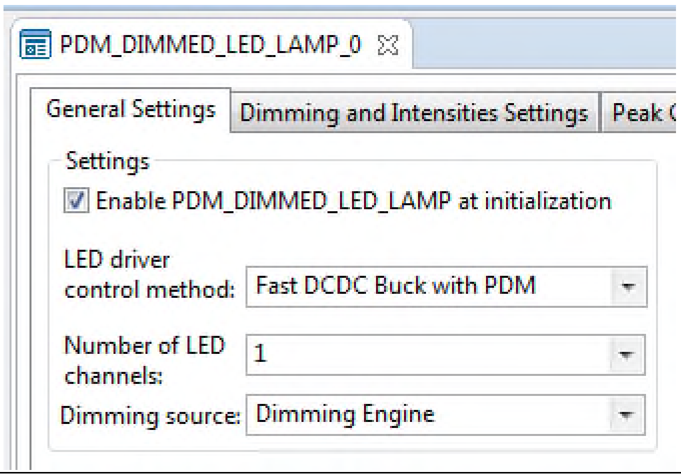
4. CCU8 off-time
   To change the CCU8 off-time:
```
XMC_CCU8_SLICE_SetTimerCompareMatch(CCU80_CC81, XMC_CCU8_SLICE_COMPARE_CHANNEL_1,
NEW_OFFTIME);
```

To effect the change:
```
XMC_CCU8_EnableShadowTransfer(CCU80, XMC_CCU8_SHADOW_TRANSFER_SLICE_1);
```

3.4.2 Implementation with DAVE™ APPs

This section describes how the PDM_DIMMED_LED_LAMP APP in DAVE™ can be used to configure the resources required to implement a CCM buck LED driver as shown in Figure 40.

In the UI of the PDM_DIMMED_LED_LAMP APP (Figure 41), the options for the LED driver control method that are available for the selected microcontroller are listed in the drop-down box under "LED driver control method". These options correspond to the ACMP output to CCUx slice routing options as summarized by Table 5.



Settings
☑ Enable PDM_DIMMED_LED_LAMP at initialization
LED driver control method: Fast DCDC Buck with PDM
Number of LED channels: 1
Dimming source: Dimming Engine

Figure 41 Configurations: PDM_DIMMED_LED_LAMP APP (general settings)

Table 5 LED driver control method

APP UI Options	Corresponding routing option number
Direct PDM (External LED Drivers)	-
Slow DCDC buck with PDM	1
Simple DCDC buck with PDM	2
Fast DCDC buck with PDM	3

By selecting "Fast DCDC buck with PDM", the APPs corresponding to the required hardware resources are aggregated into the project (Figure 42). The necessary configurations for the resources have also been automatically configured. Optional configurations can be carried out manually in the UI of the respective APPs.

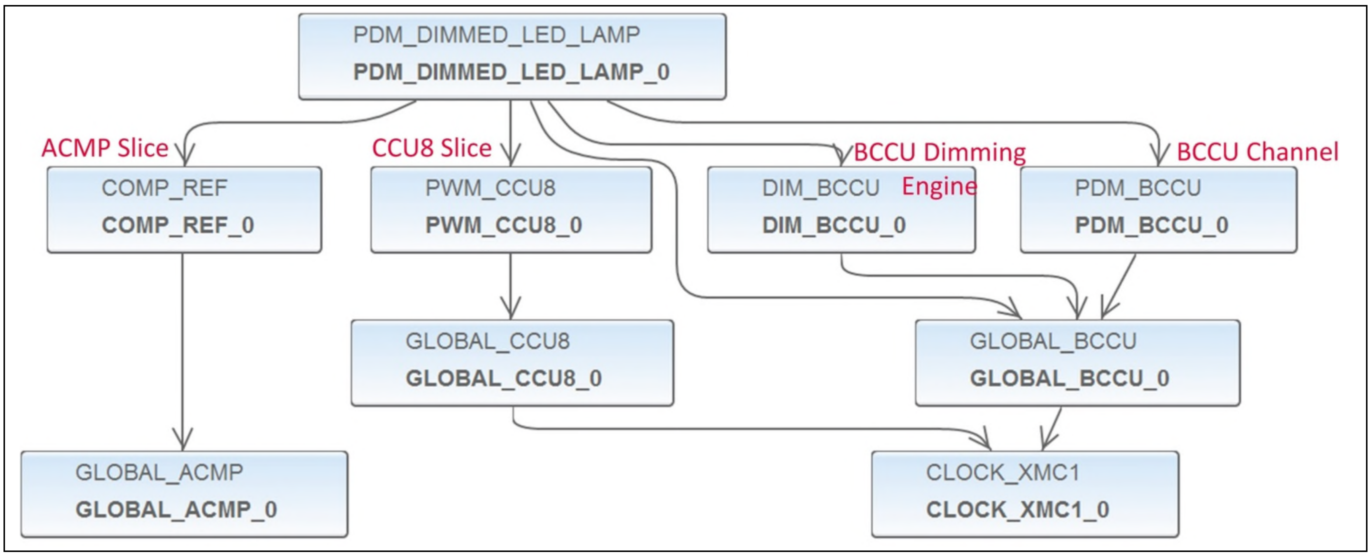
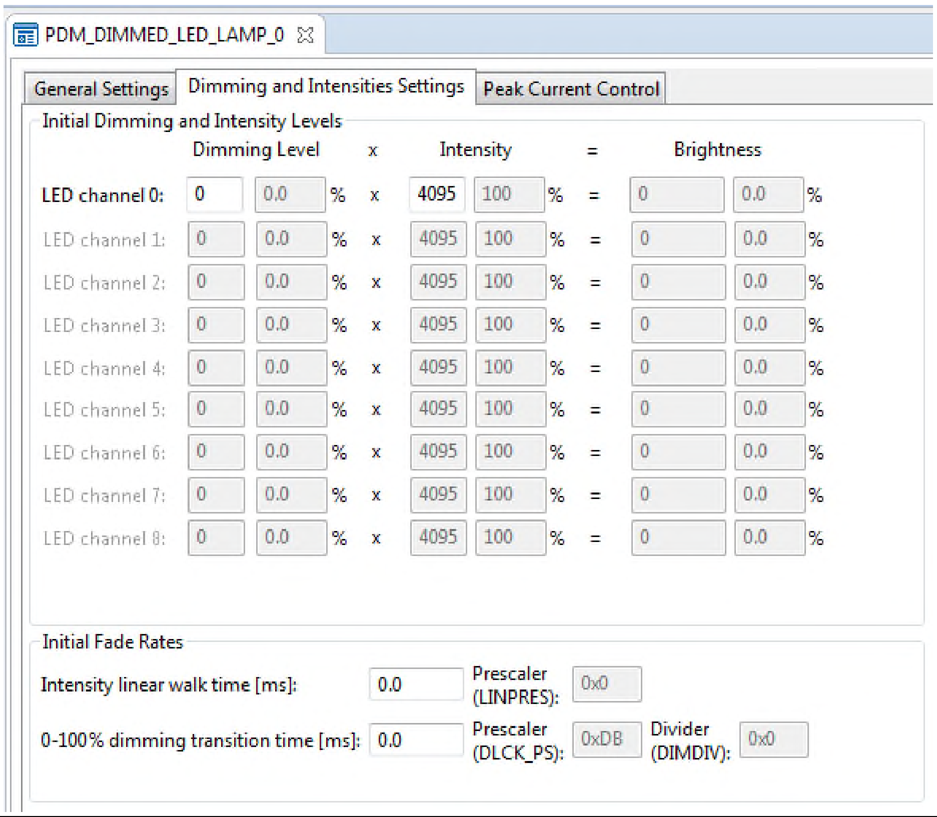


Figure 42 APPs corresponding to the required hardware resources

Under the "Dimming and Intensities Settings" tab in the UI, the initial dimming level, intensity value, linear walk time and dimming transition time can be configured (Figure 43).



Initial Dimming and Intensity Levels

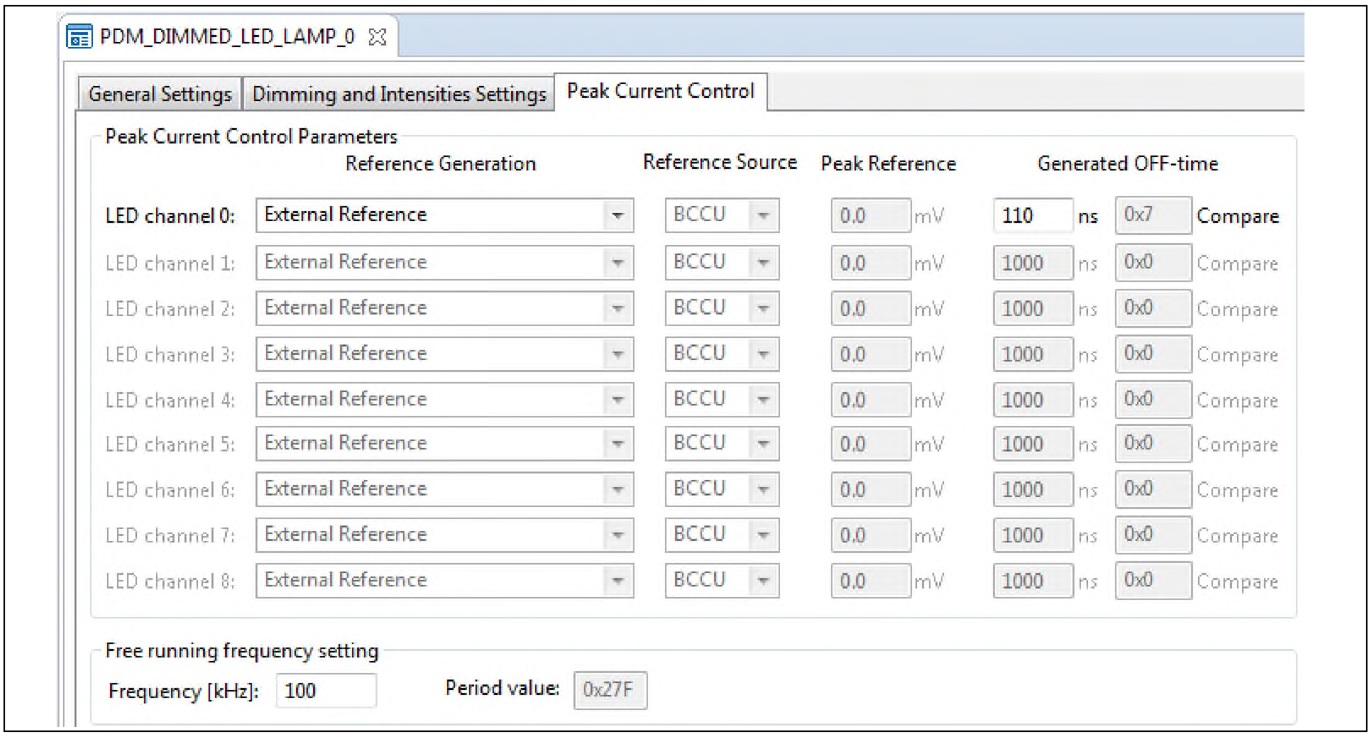
	Dimming Level			x	Intensity			=	Brightness		
LED channel 0:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 1:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 2:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 3:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 4:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 5:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 6:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 7:	0	0.0	%	x	4095	100	%	=	0	0.0	%
LED channel 8:	0	0.0	%	x	4095	100	%	=	0	0.0	%

Initial Fade Rates

Intensity linear walk time [ms]:	0.0	Prescaler (LINPRES):	0x0		
0-100% dimming transition time [ms]:	0.0	Prescaler (DLCK_PS):	0xDB	Divider (DIMDIV):	0x0

Figure 43 Configurations: PDM_DIMMED_LAMP_APP (Dimming and Intensities Settings)

Under the "Peak Current Control" tab in the UI, the ACMP reference generation option and desired generated off-time can be configured (Figure 44).



PDM_DIMMED_LED_LAMP_0

General Settings | Dimming and Intensities Settings | Peak Current Control

Peak Current Control Parameters

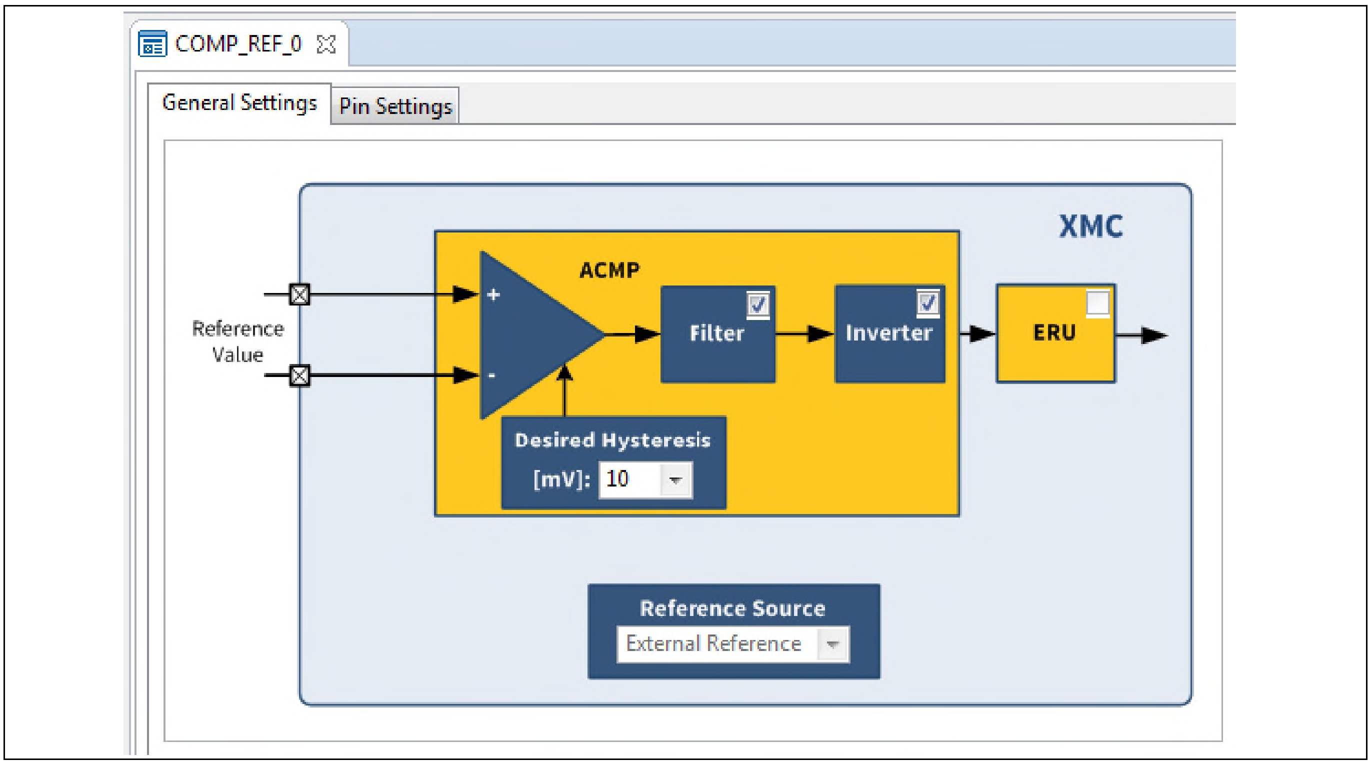
	Reference Generation	Reference Source	Peak Reference		Generated OFF-time			
LED channel 0:	External Reference	BCCU	0.0	mV	110	ns	0x7	Compare
LED channel 1:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 2:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 3:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 4:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 5:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 6:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 7:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare
LED channel 8:	External Reference	BCCU	0.0	mV	1000	ns	0x0	Compare

Free running frequency setting

Frequency [kHz]: 100 Period value: 0x27F

Figure 44 Configurations: PDM_DIMMED_LED_LAMP APP (Peak Current Control)

The ACMP slice output can be configured via COMP_REF APP UI (Figure 45).

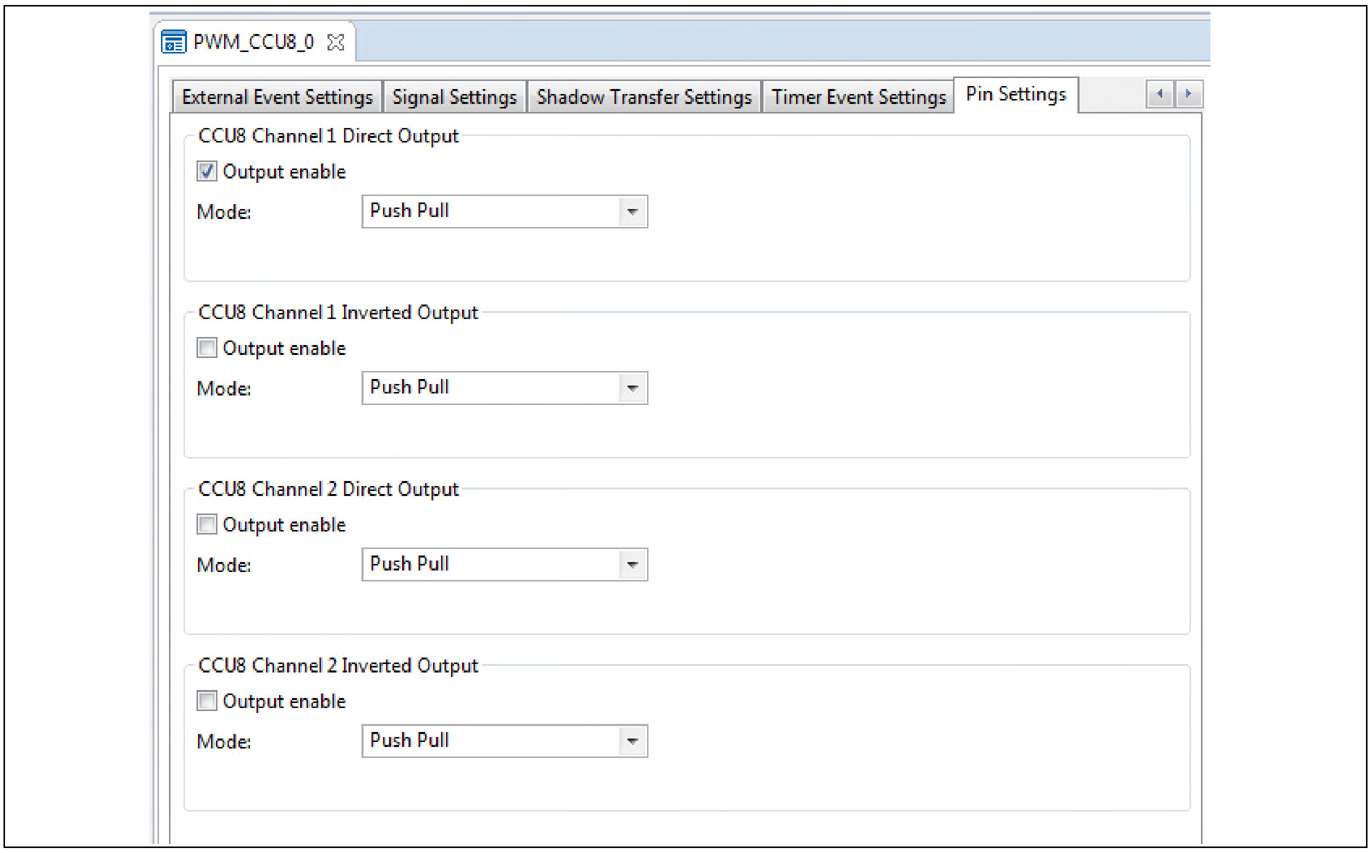


COMP_REF_0

General Settings | Pin Settings

Figure 45 Configurations: COMP_REF APP

The CCU8 slice output pin can be selected in the PWM_CCU8 APP UI under “Pin Settings” tab (Figure 46).



CCU8 Channel 1 Direct Output
☑ Output enable
Mode: Push Pull

CCU8 Channel 1 Inverted Output
☐ Output enable
Mode: Push Pull

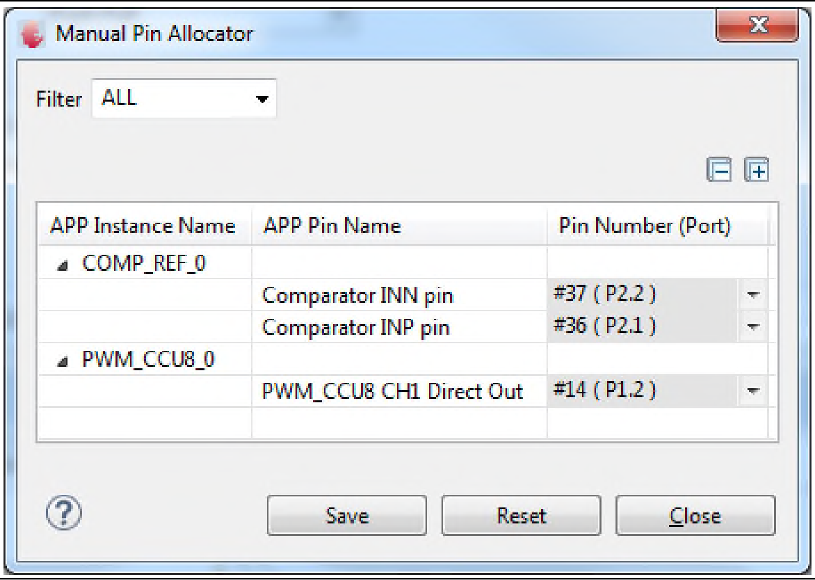
CCU8 Channel 2 Direct Output
☐ Output enable
Mode: Push Pull

CCU8 Channel 2 Inverted Output
☐ Output enable
Mode: Push Pull

Figure 46 Configurations: PWM_CCU8 APP (Pin settings)

Pin assignment

The input pins to the ACMP slice and the output pin for the CCU8 slice can be assigned using the "Manual pin allocator" (Figure 47).



Filter: ALL

APP Instance Name	APP Pin Name	Pin Number (Port)
COMP_REF_0		
	Comparator INN pin	#37 (P2.2)
	Comparator INP pin	#36 (P2.1)
PWM_CCU8_0		
	PWM_CCU8 CH1 Direct Out	#14 (P1.2)

Figure 47 Pin assignment for XMC™ LED current control explorer

Application code

This section provides a guide to adjust the dimming level and color of the LED lamp, the ACMP reference and the CCU8 off-time in-application.

1. Dimming level

   The DAVE™-generated configuration data structure, *PDM_DIMMED_LED_LAMP_0.config*, for the PDM_DIMMED_LED_LAMP_0 APP instance also holds the dim level value of the dimming engine used in the lamp. It acts like a shadow register for the respective register bit field **DLSz.TDLEV**.

   As an example, to change the dim level to 100%:

```
PDM_DIMMED_LED_LAMP_0_config.dim_level = 4095;
```

   The line above sets the dimming engine target level for the desired dim level. To effect the brightness change, the dimming process has to be started:

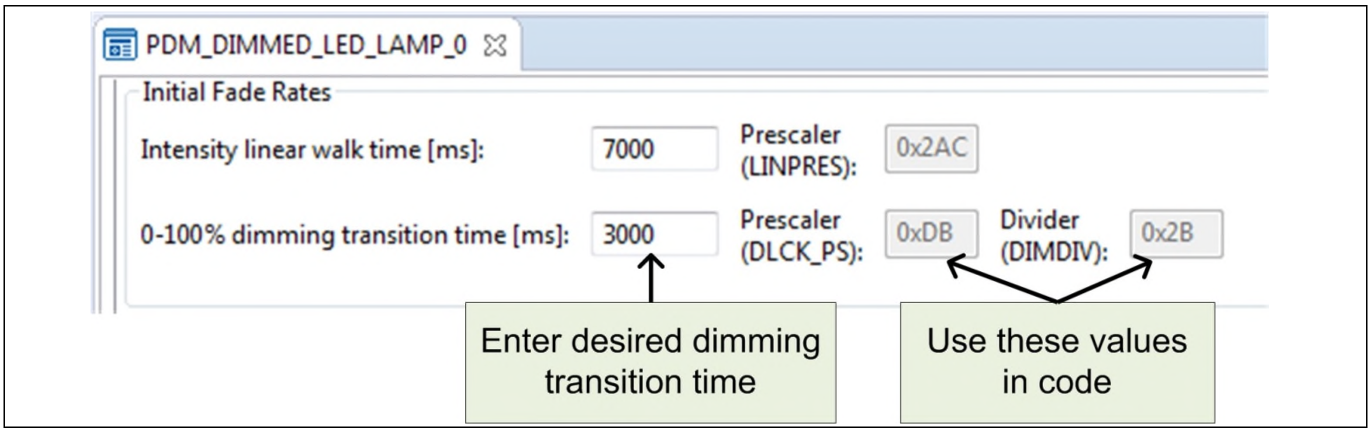
```
PDM_DIMMED_LED_LAMP_SetDimLevelExponential(&PDM_DIMMED_LED_LAMP_0);
```

   This function programs the BCCU register bit fields with the configured dimming transition time, the target dimming level and starts the dimming process. The dimming transition time used is configured the same way as previously in the UI of the PDM_DIMMED_LED_LAMP APP.

   To use a different dimming transition time for changing lamp brightness:

```
/* Configure dimming transition time = 3s, and start dimming process */
PDM_DIMMED_LED_LAMP_SetDimLevelExponentialAdv(&PDM_DIMMED_LED_LAMP_0, 0x2B, 0xDB);
```

   The PDM_DIMMED_LED_LAMP APP UI can be used to easily calculate the prescaler and divider values for the desired dimming transition time (Figure 48).


PDM_DIMMED_LED_LAMP_0

Initial Fade Rates

Intensity linear walk time [ms]: 7000 Prescaler (LINPRES): 0x2AC

0-100% dimming transition time [ms]: 3000 Prescaler (DLCK_PS): 0xDB Divider (DIMDIV): 0x2B

Figure 48 Use PDM_DIMMED_LED_LAMP APP to calculate dimming transition time prescaler and divider

2. Color

PDM_DIMMED_LED_LAMP_0.config also holds the intensity values of the respective BCCU channels used in the lamp and also acts as a shadow register for the respective register bit fields INTSy.TCHINT.

As an example, to change the light color to red:

```
PDM_DIMMED_LED_LAMP_0.config->led_intensity[0] = 4095;
PDM_DIMMED_LED_LAMP_0.config->led_intensity[1] = 0;
PDM_DIMMED_LED_LAMP_0.config->led_intensity[2] = 0;
```

The lines above set the respective channel intensities for the desired color. To effect the color change, the linear walk has to be started:

```
PDM_DIMMED_LED_LAMP_SetColor(&PDM_DIMMED_LED_LAMP_0);
```

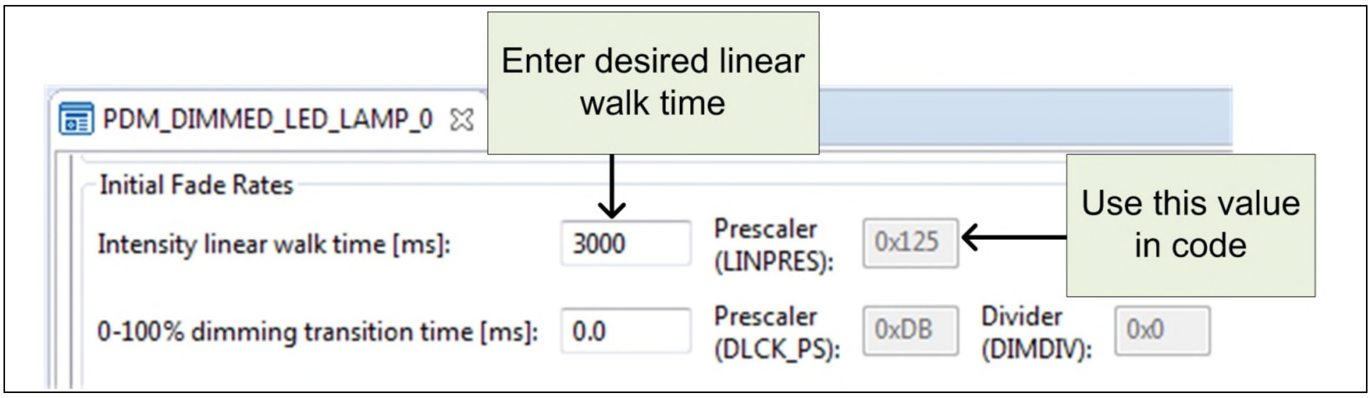
This function programs the BCCU register bit fields with the requested intensity values and then synchronizes the start of the linear walk for all channels of the lamp, to ensure a smooth and natural color transition. The linear walk time is configured the same as previously in the UI of the PDM_DIMMED_LED_LAMP APP.

The combination of the channel intensities results in a color. One simple color scheme that can be followed is as suggested inTable 4.

To use a different linear walk time for changing lamp color:

```
PDM_DIMMED_LED_LAMP_SetColorAdv(&PDM_DIMMED_LED_LAMP_0, 0x125); /* walk time = 3s */
```

The PDM_DIMMED_LED_LAMP APP UI can be used to easily calculate the prescaler for the desired walk time (Figure 49).



Initial Fade Rates

Intensity linear walk time [ms]: 3000 Prescaler (LINPRES): 0x125

0-100% dimming transition time [ms]: 0.0 Prescaler (DLCK_PS): 0xDB Divider (DIMDIV): 0x0

Figure 49 Use PDM_DIMMED_LED_LAMP APP to calculate linear walk time prescaler

3. ACMP reference level (applies only if internal reference generation is used)

   PDM_DIMMED_LED_LAMP_0.config also holds the peak current control reference value. As an example, to change the reference value of the first LED channel to 575 mV:

```
PDM_DIMMED_LED_LAMP_0.config->peak_cur_ctrl_refval[0] = 575U;
```

To effect the change, call the following function:

```
PDM_DIMMED_LED_LAMP_SetPeakReference(&PDM_DIMMED_LED_LAMP_0);
```

4. Generated off-time
   PDM_DIMMED_LED_LAMP_0.config also holds the generated off-time value. As an example, to change the off-time value of the first LED channel to 200 ns:

   ```
   PDM_DIMMED_LED_LAMP_0.config->peakcur_ctrl_offtime[0] = 200U;
   ```

   To effect the change, call the following function:

   ```
   PDM_DIMMED_LED_LAMP_SetOffTime(&PDM_DIMMED_LED_LAMP_0);
   ```

3.5 Tuning LED current ripple

In this section, a step-by-step guide is provided to tune the LED current ripple. It is assumed that the software project has been created beforehand, according to 3.4.1 Implementation using XMC™ lib or 3.4.2 Implementation with DAVE™ APPs.

1. Initialize the following parameters to safe values as shown in Table 6.

Table 6 Safe parameters

Parameter	Value
BCCU FCLK	0.8 MHz (0x50)
Generated off-time	1000 ns (0x40)
Dimming level	40 (1 %)

2. Run the project with the targeted LED engine connected to the board.
3. Observe the LED current waveform on an oscilloscope. Measure the following:
   a. LED current rise time
   b. LED current fall time
   c. Required off-time
   Refer to Figure 50 for an example of the measurements.

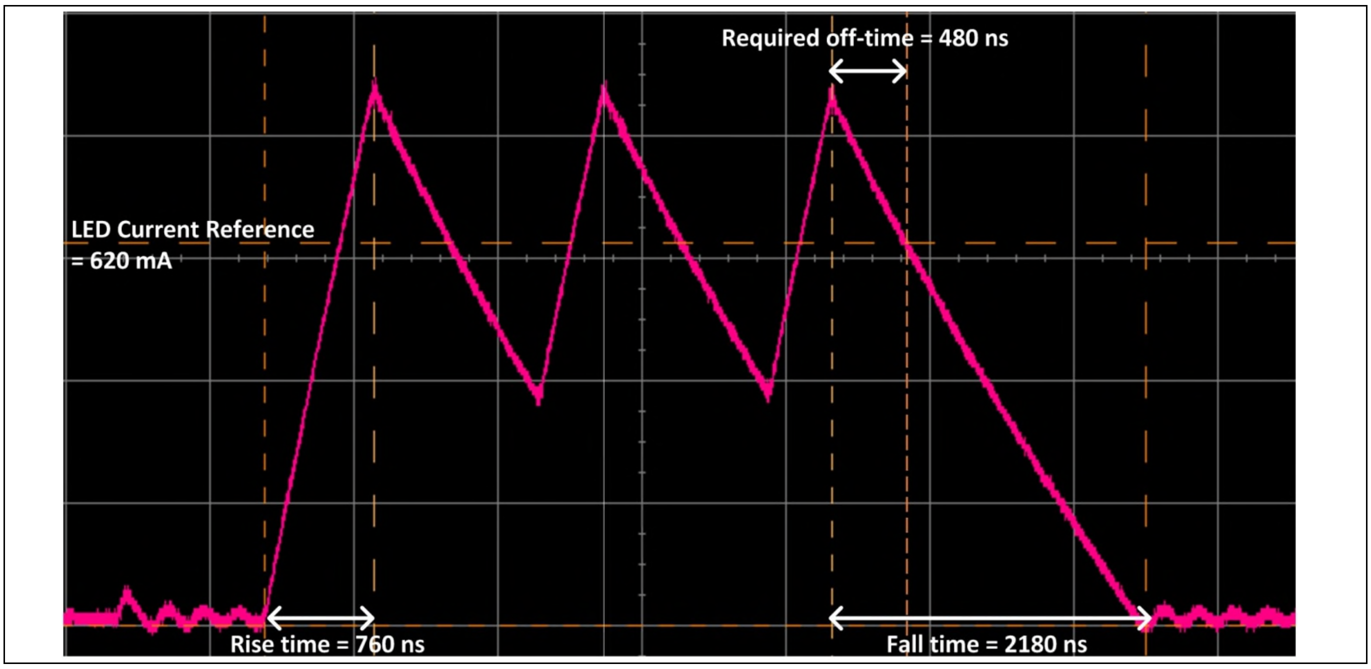


Figure 50 LED current waveform and the required measurements

4. Change the frequency of BCCU FCLK such that the bit-time is the same as the LED current rise time or fall time, whichever is longer. This is to ensure that there is enough time for the LED current to rise to the desired peak level during a BCCU on-bit and enough time for the LED current to drop to zero during a BCCU off-bit.

   In the example shown in Figure 50, the fall time is longer. The required BCCU FCLK can be calculated as follows:

$$
\begin{aligned}
FCLK &= \frac{1}{bit\ time} \times 4 \\
&= \frac{1}{2180 \times 10^{-9}} \times 4 \\
&\approx 1.835\ MHz
\end{aligned}
$$

5. Change the generated off-time based on the value measured previously. This is to ensure that there is enough time for the LED current to drop to just below the reference level.

   In the example shown in Figure 50, the measured off-time is 480 ns.
6. With the new values, run the project again to confirm that the new values are satisfactory. Figure 51 shows the LED current ripple after successful tuning. Additional iterations of tuning may be required to achieve an optimized state.

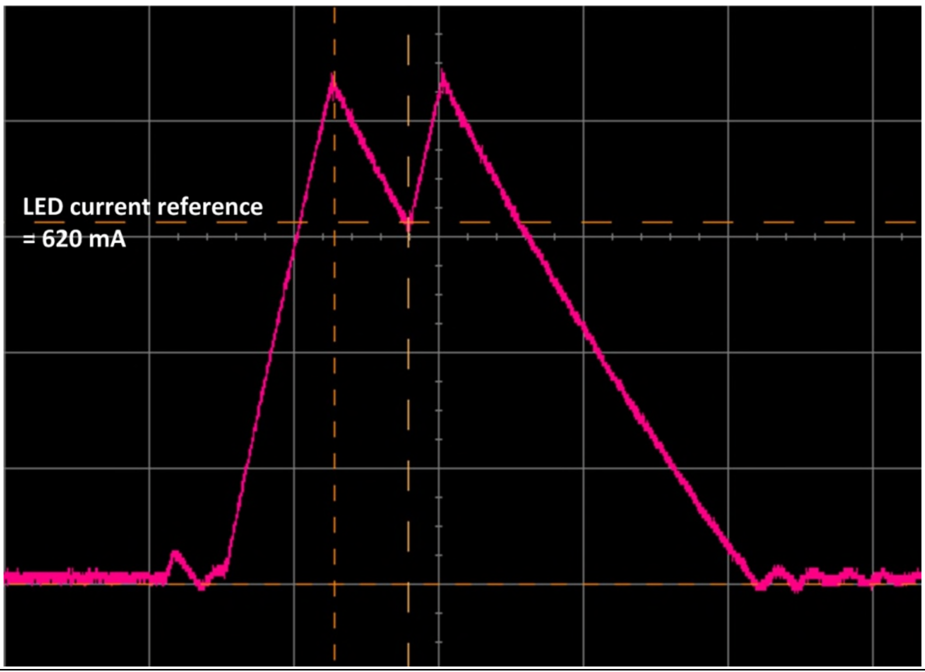


Figure 51 Optimized LED current ripple waveform

Revision history

Major changes since the last revision

Page or reference	Description of change

Edition 2016-10-15
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
AN_201609_PL30_028