Tunable white LED lamp control with RGB LED lighting shield

XMC1000

About this document

Scope and purpose

This document provides hints for using the RGB LED lighting shield to drive and control tunable white LEDs.

Intended audience

This document is intended for engineers interested in evaluating or making prototypes for their tunable white LED lamp applications.

Applicable products

- XMC1202
- DAVE™

References

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

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

Infineon: RGB LED lighting shield, http://www.infineon.com/arduino

Table of contents

1 Introduction

A tunable white LED consists of two LED channels: the cool white channel and warm white channel (Figure 1). The control parameters required for tunable white LEDs are brightness and color temperature. These parameters are typically controlled via a dimmer switch.

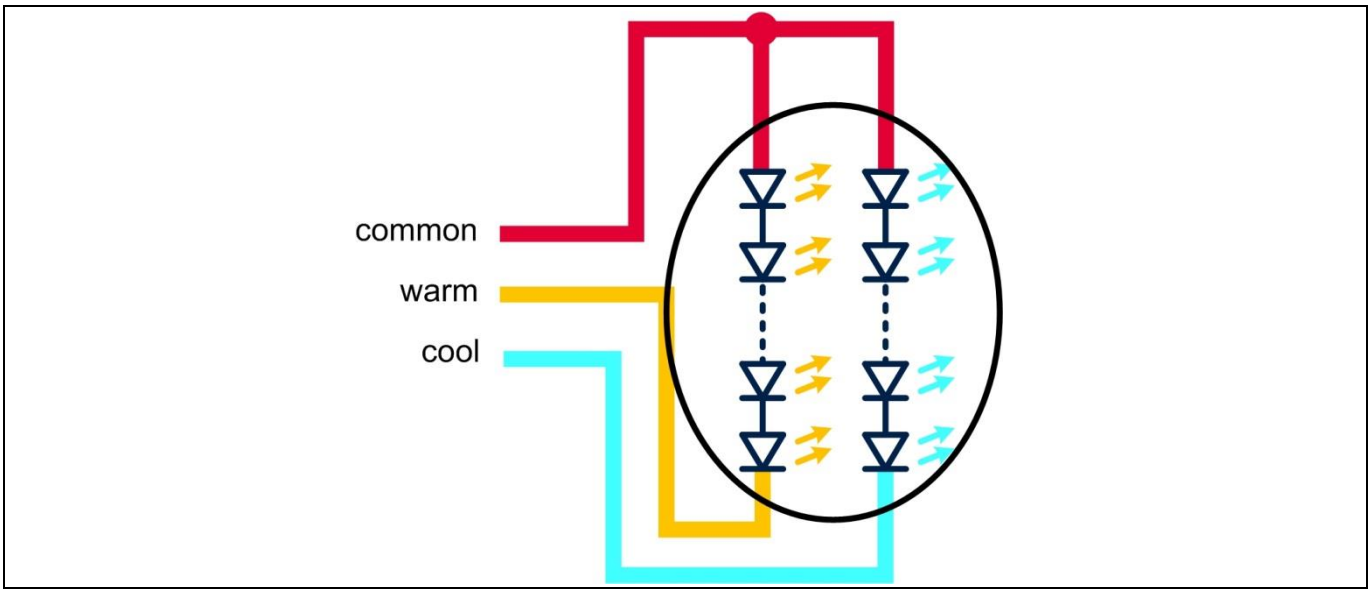

Figure 1 Tunable white LED lamp

The RGB LED lighting shield can be used for evaluating or creating rapid prototypes for tunable white LED applications. The optional DMX512 interface can be used in place of a dimmer switch to control the tunable white LEDs. Alternatively, potentiometers can be used for a simpler and more cost effective method of evaluation.

This document provides hints for modifying the RGB LED lighting shield to drive the tunable white LEDs and control them via potentiometers (Figure 2). The supplied application code is developed with Everlight's CHI3030, which has a forward voltage of 25.5 V for both warm and cool channels at 500 mA. Other light engines with similar characteristics can also be used directly with the code. Figure 3 shows the lamp turned on with warm white colour (temperature: 2700 K). Figure 4 shows the lamp turned on with cold white colour (temperature: 5700 K).

The hardware and software presented in this document can control the lamp's color temperature from 2700 K to 5700 K in 4095 discrete steps. It can also dim the intensity of light in 4095 discrete steps, with flicker-free quality light down to 0.5% dimming level.

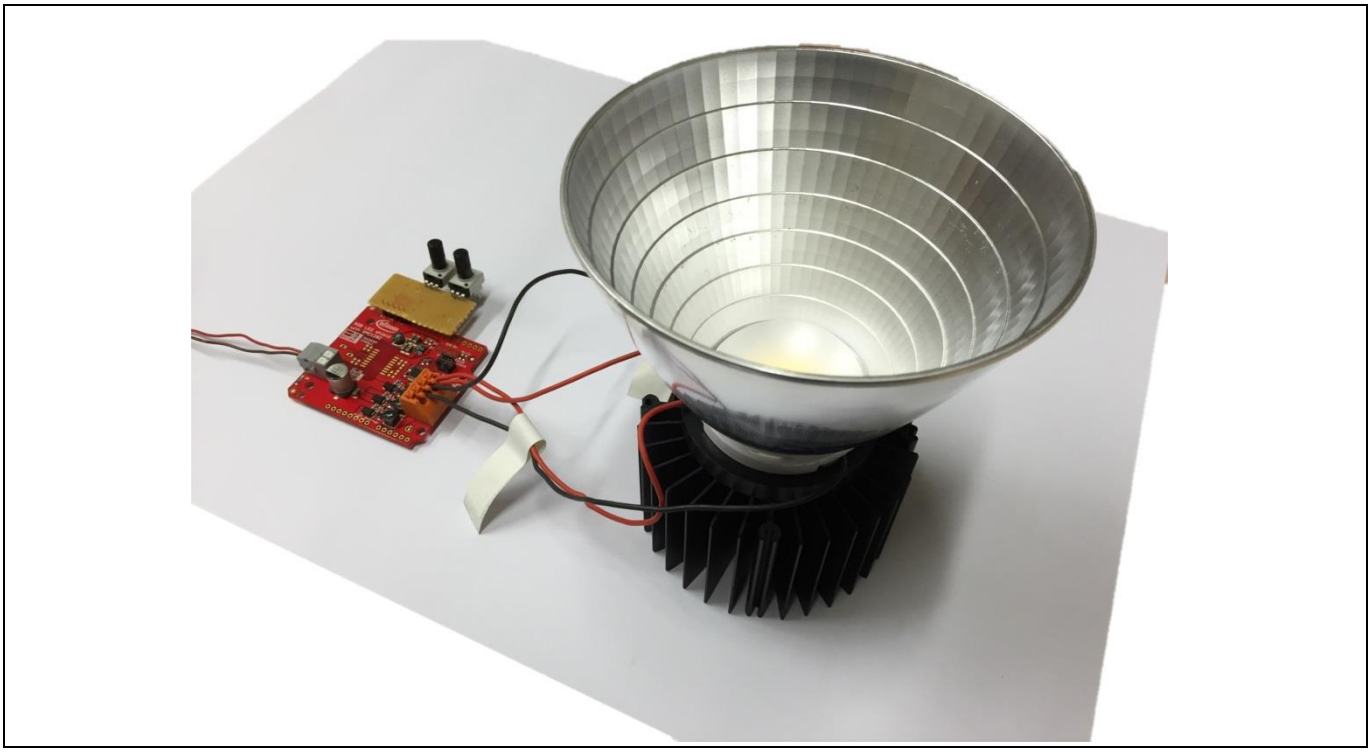

Figure 2 Modified RGB LED lighting shield for controlling tunable white LED lamp (in picture: Everlight's CHI3030)

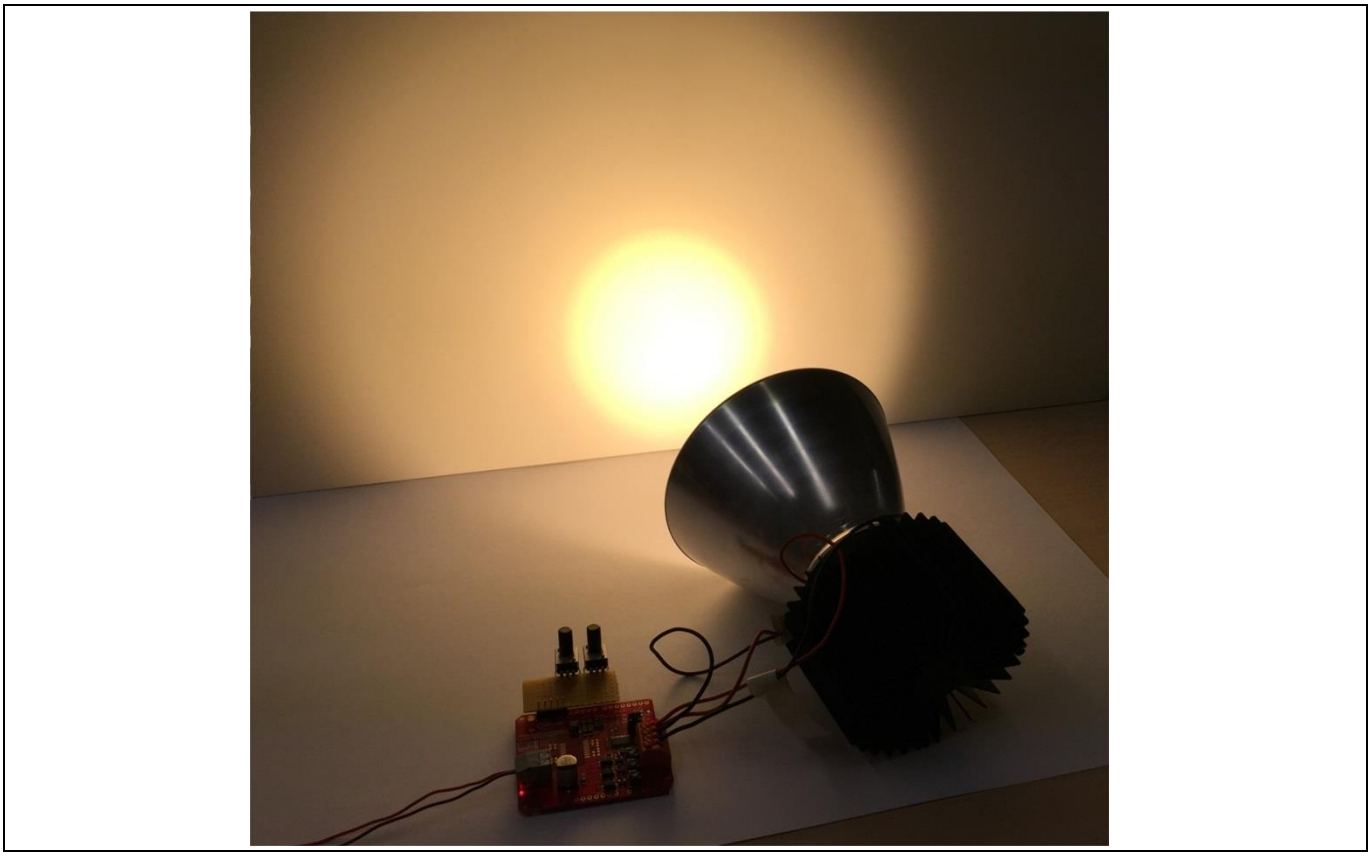

Figure 3 Warm white light (color temperature: 2700 K)

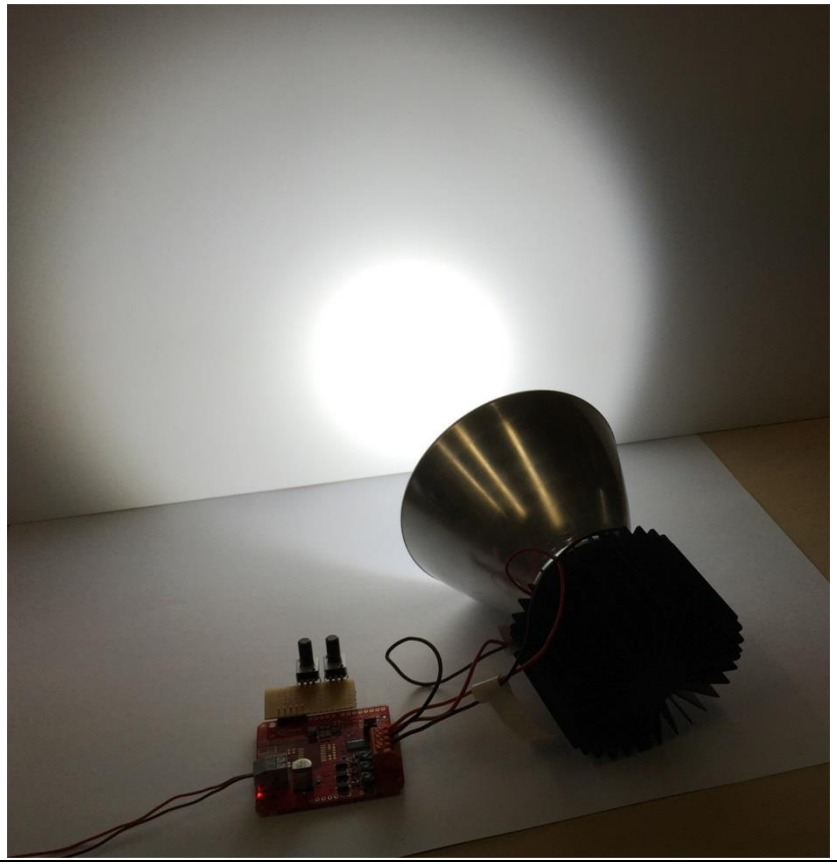

Figure 4 Cool white light (color temperature: 5700 K)

2 Hardware

This section describes how the potentiometers and the tunable white LED lamp can be connected to the RGB LED lighting shield.

2.1 Connecting potentiomers

Two potentiometers are used – one to control brightness and another to control color temperature. This means that two ADC channels are required. P2.10 and P2.11, which are initially reserved for IIC communication (SDA and SCL respectively), can be used.

Note: Therefore, IIC communication is not available when using the pins for ADC measurement.

2.1.1 Schematics

Figure 5 provides an illustration of the connections from the potentiometers to the XMC1202.

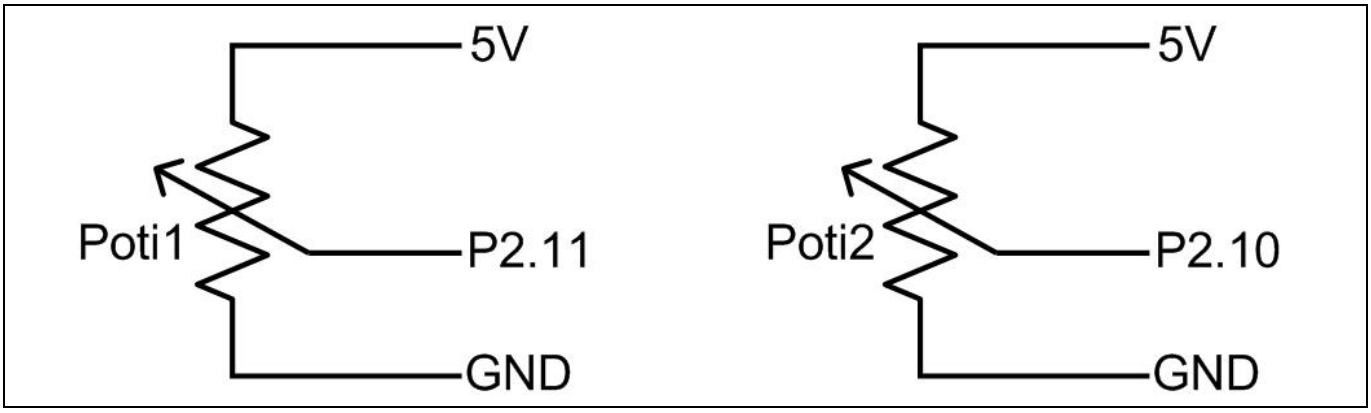

Figure 5 Schematics for potentiometer connections

2.1.2 Hardware modifications

Five of the Arduino™ connector pins, located on the top right-hand corner of the board are used for connecting to the potentiometers. Table 1 lists the pins used and their new functions. Figure 6 illustrates the connections between the potentiometers and the RGB LED lighting shield.

Table 1 Arduino™ connector pin function changes

Old pin function	New pin function
SCL	Poti 1 output
SDA	Poti 2 output
AREF	5V
GND	GND
13	GND

The pull-up resistors connected to the SCL and SDA pins must be removed (Figure 7) to prevent cross-talk between the potentiometers.

The AREF pin must be connected to 5 V. This connection can be made using a piece of wire on the bottom side of the shield, as shown in Figure 8.

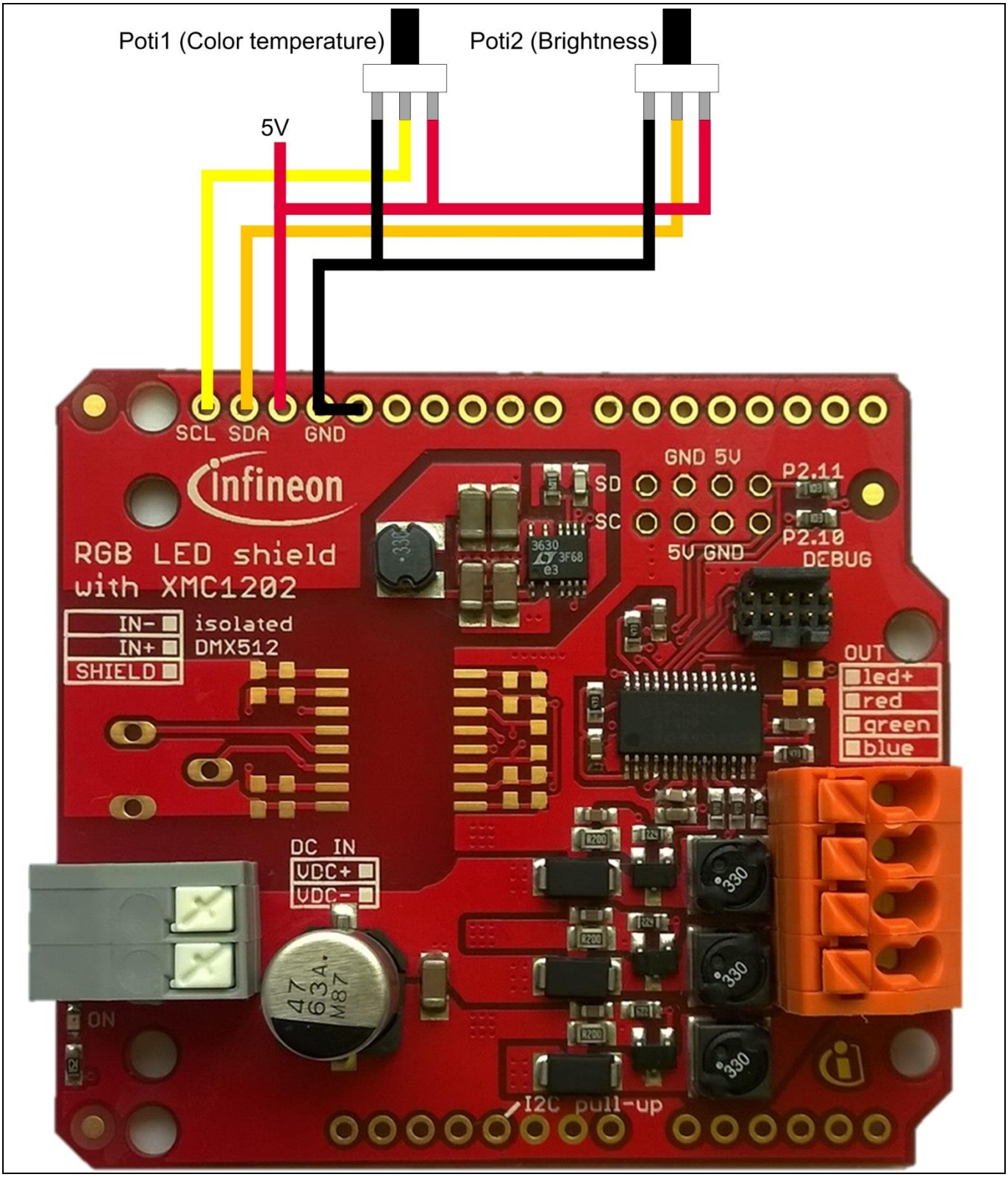

Figure 6 Connecting potentiometers to the RGB LED lighting shield

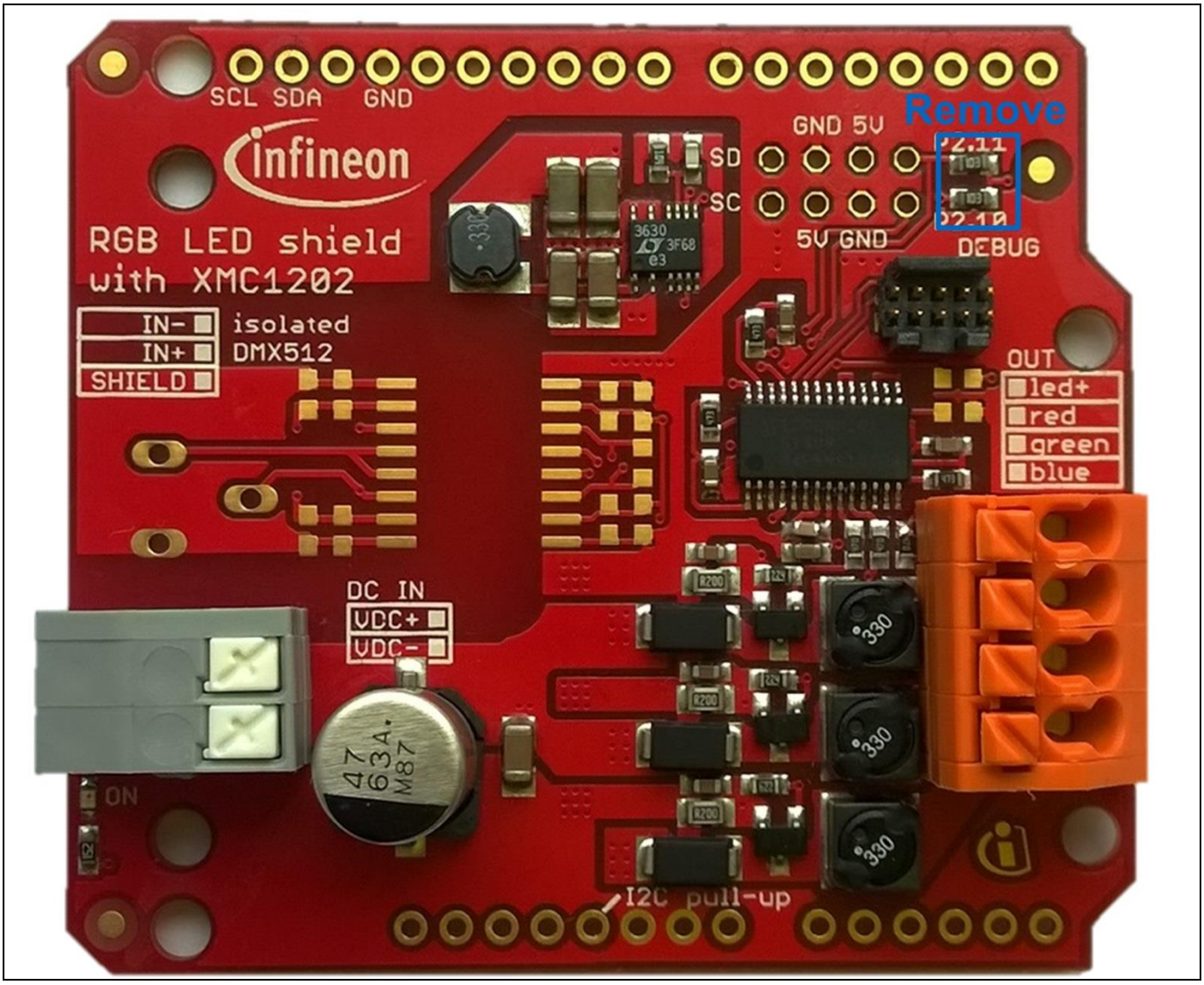

Figure 7 IIC pull-up resistors to be removed

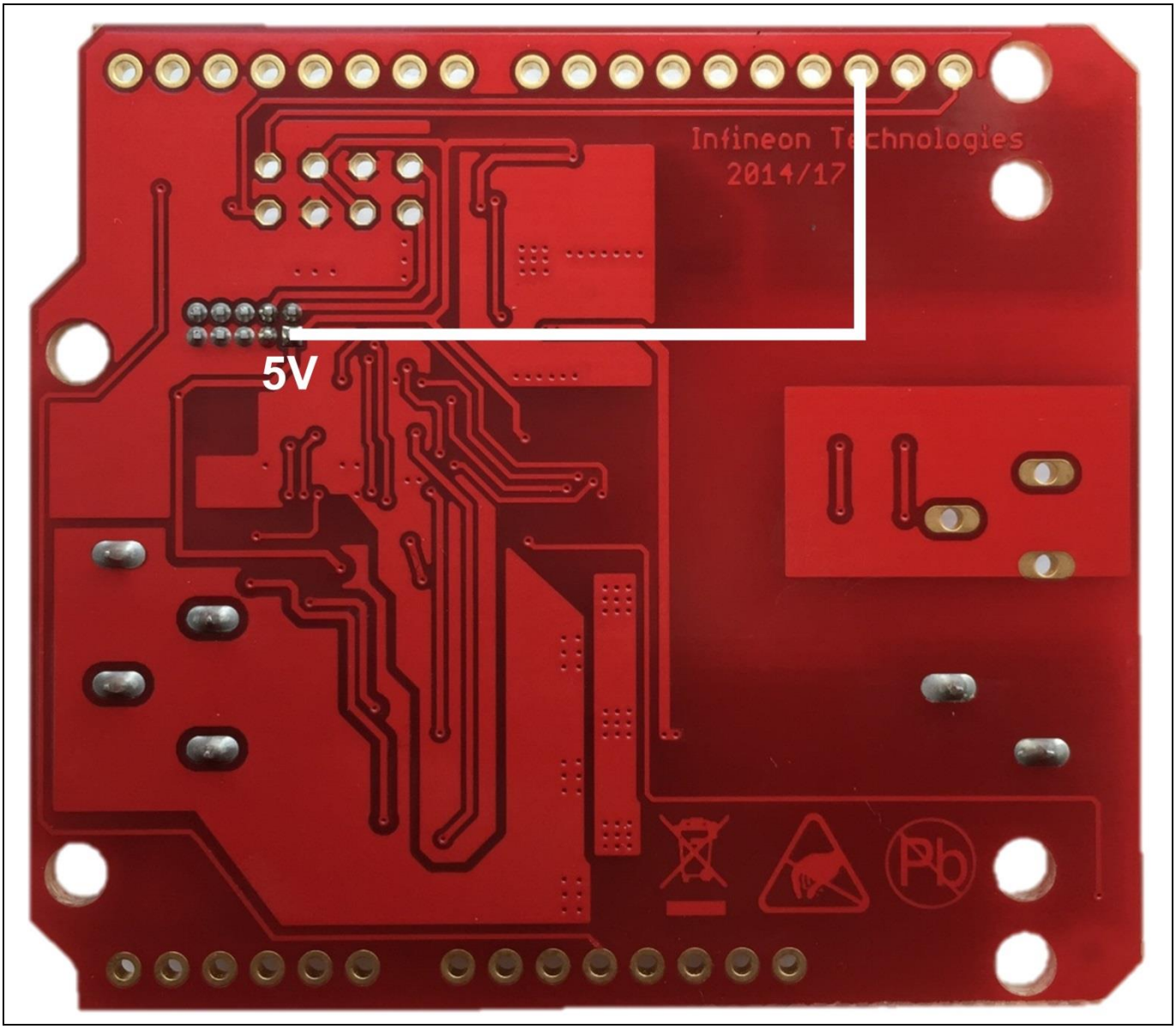

Figure 8 Connecting AREF pin to 5 V

2.2 Connecting the tunable white LED lamp

Two out of the three available MOSFET gate drive outputs are used to drive a tunable white LED lamp. Figure 9 illustrates the connections between the tunable white LED lamp and the RGB LED lighting shield.

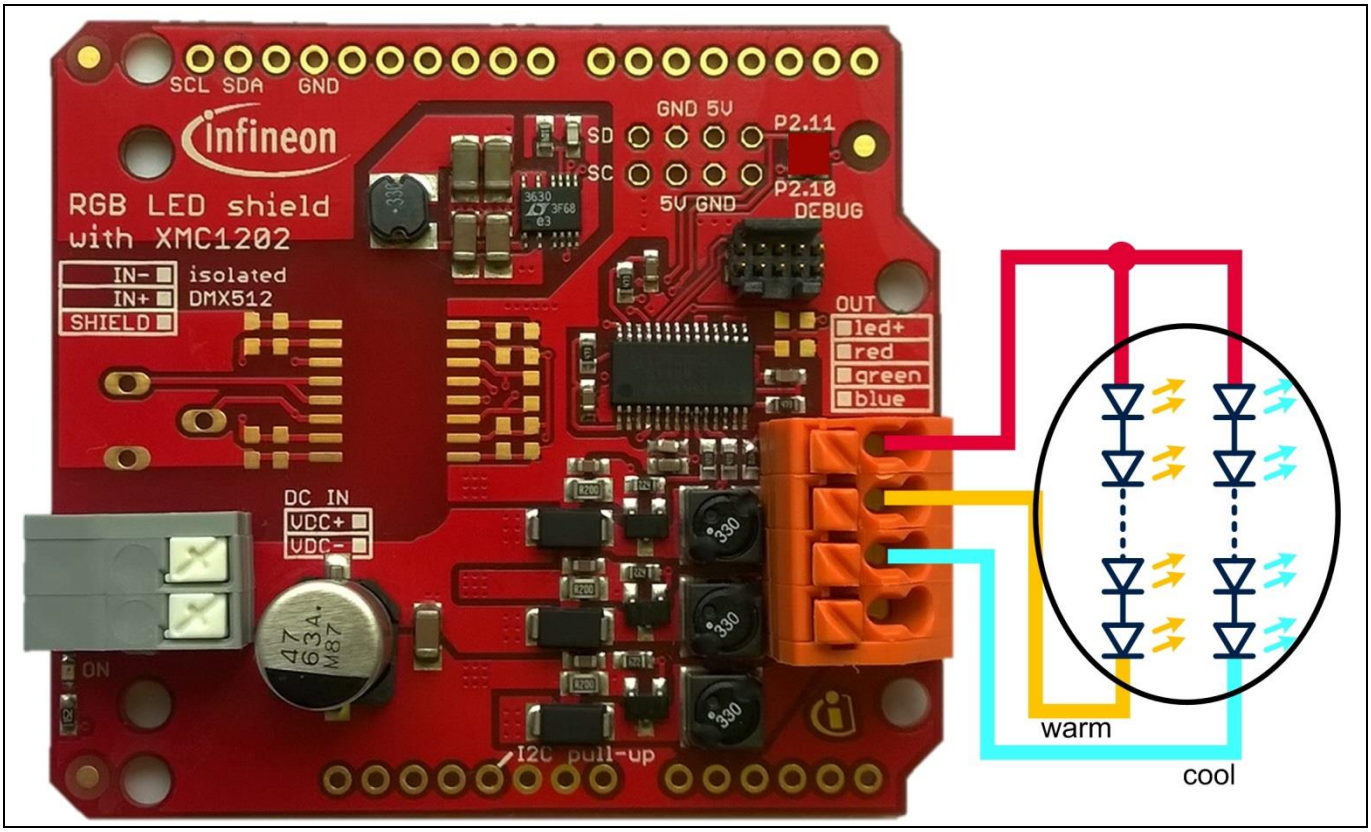

Figure 9 Connecting the tunable white LED lamp to the shield

3 Software

This section describes the additional software required to use the potentiometers. The software is written based on the XMC™ library.

3.1 ADC measurement

The ADC conversions of the 2 ADC channels are requested via the ADC group scan source. Pins P2.10 and P2.11 are used so channels 2 and 3 of group 1 are measured. Alternatively, channels 3 and 4 of group 0 can be measured.

The ADC conversion requests are triggered by the BCCU PDM outputs. The respective BCCU channels trigger the ADC on the falling edge using trigger mode 0 with a quarter bit time delay. This configuration minimizes the noise on the ADC input pin when the measurement is made.

The ADC measurement result is stored as a 12-bit value without any accumulation. The request source event interrupt is enabled. This is triggered when both ADC channels have been converted.

3.2 Result processing

In the request source event interrupt service routine (*VADC0_G1_0_IRQHandler*), the conversion results are put through a low-pass filter (Figure 10) to smooth the jitter.

$$Result_{filtered} = \frac{Result_{current} + Result_{average} - \left(\frac{Result_{average}}{2^8}\right)}{2^{12}}$$

Figure 10 Equation for low-pass filter used

The filtered results from both potentiometers are then used with look-up tables (LUTs) to get the final values. Two separate LUTs are used in this project – one for the dimming level and one for the color temperature. The LUT for dimming level is called *expdim* and can be found in the header file, *expdim.h*. This LUT contains 256 dimming level values, incremented following an exponential profile. The LUT for the color temperature is called *lintemp* and can be found in the header file, *lintemp.h*. This LUT contains 256 channel intensity values, incremented in a linear manner.

3.3 Brightness control

The final dim level value extracted from the LUT *expdim* is programmed as the target dim level value for the BCCU dimming engine. The dimming process is then started with fade rate set as 0. This configures the dimming transition to take place immediately. A different fade rate can be configured by programming a non-zero value to the DIMDIV bit-field before starting the dimming process.

3.4 Color temperature control

The final color temperature value extracted for the LUT *lintemp* is programmed as the channel intensity for the warm white LED channel. A simple color temperature scheme is used in this project, whereby the intensities of the warm and cool white LED channels should add up to 4095. The required channel intensity for the cool white LED channel is calculated and programmed. To maintain a smooth color temperature transition, the linear walk is started concurrently in both channels. A short linear walk time of 48 ms is set as default in the project.

This can be changed by programming the linear walk prescaler value for both channels before starting the linear walk.

4 Programming the shield

This section provides a step-by-step guide to get the RGB LED lighting shield programmed and running.

4.1 DAVE™ Integrated Development Environment (IDE)

DAVE™ is a free Eclipse based IDE including GNU C-compiler, debugger, comprehensive code repository, hardware resource management, and code generation plug-in. DAVE™ is required for downloading the code to the XMC1202 microcontroller on the RGB LED lighting shield.

4.1.1 Download DAVE™

The complete package that includes the IDE, XMC™ Lib, DAVE™ APPs, EXAMPLES, and DAVE™ SDK can be downloaded from www.infineon.com/dave/v4 (Figure 11).

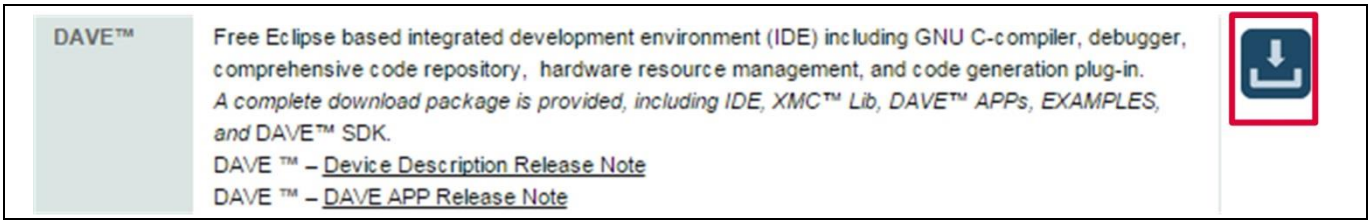

Figure 11 Download DAVE™ here

4.1.2 Install DAVE™

The step-by-step guide to install DAVE™ is available at www.infineon.com/dave/v4. There are two types of installation guides available:

1. A written guide that you can read: see the .pdf document titled “DAVE™ Quick Start” found under the “Documents” section at the above link (Figure 12).
2. A video guide that you can watch: see the video titled “DAVE™ (Version 4) Quick Start” found in the right-hand column at the above link (Figure 13).

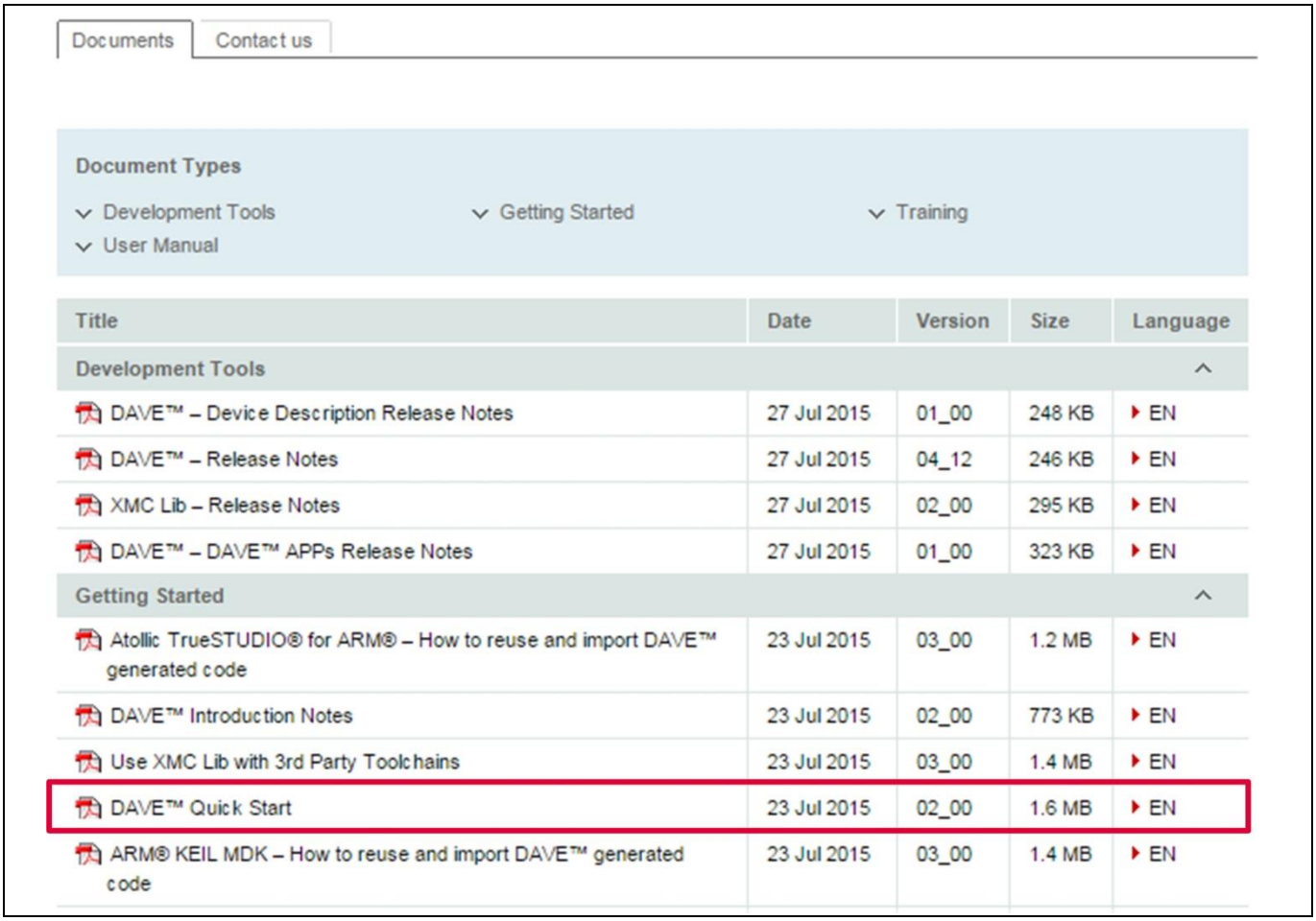

Figure 12 DAVE™ quick start document

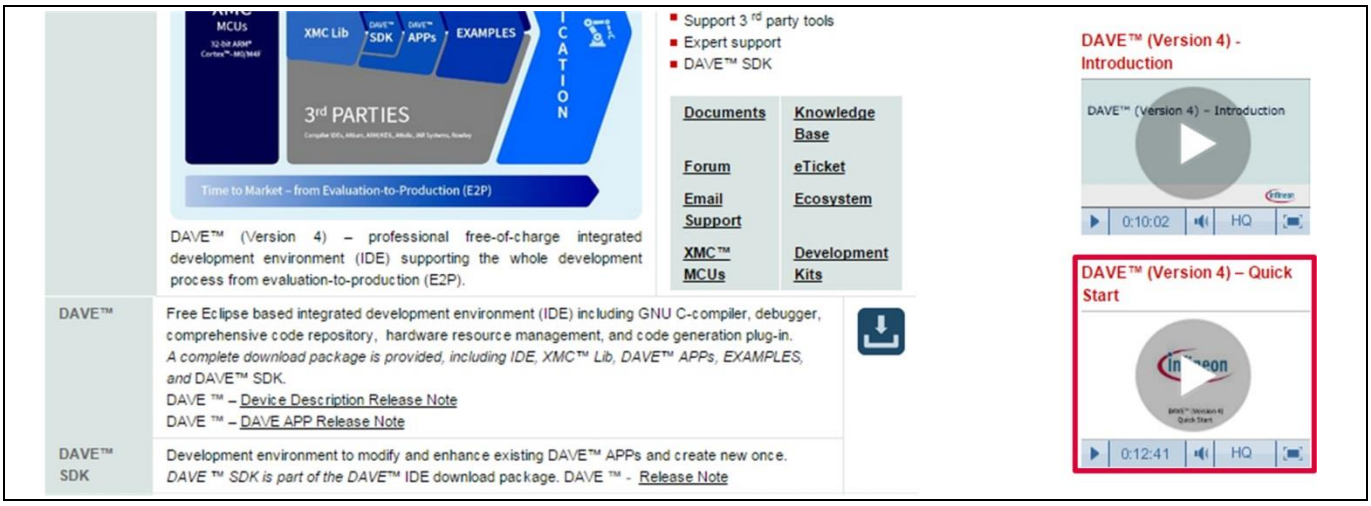

Figure 13 DAVE™ (Version 4) quick start video

Follow the instructions in the guide to install and launch DAVE™, until a new workspace is created.

4.2 Downloading the software

Follow these steps to download the software:

1. Go to www.infineon.com/arduino.
2. Click the sub-heading “RGB LED lighting shield with XMC1202 for Arduino” (Figure 14). The webpage for the RGB LED lighting shield opens in a new browser tab.

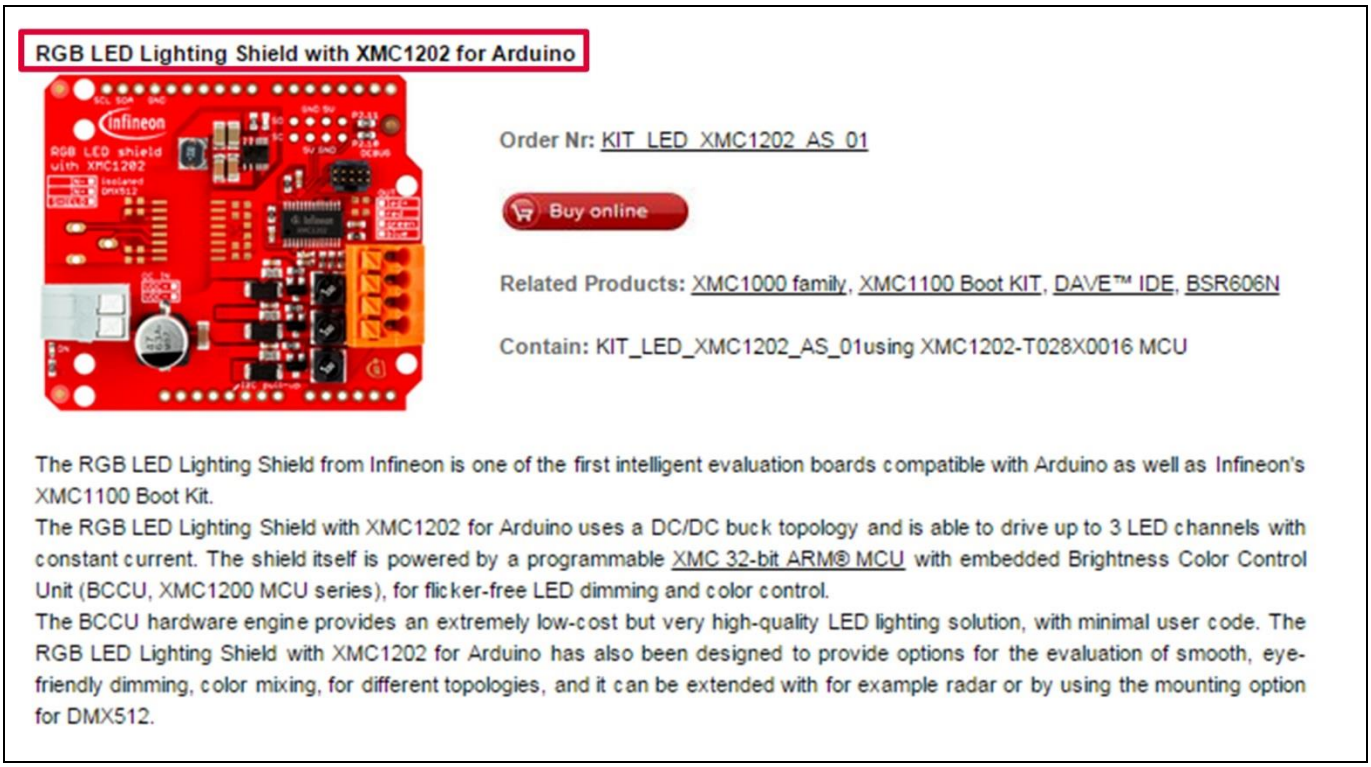

Figure 14 Click the highlighted sub-heading

3. Find the project titled “RGB LED lighting shield – tunable white LED control with potentiometers (XMC™ Lib)” under the “Application examples” section (Figure 15). Click to start download.

Figure 15 Click project name to download

4.3 Importing the downloaded software

Follow these steps to import the downloaded software into the newly created DAVE™ workspace:

1. Click File -> Import.
2. Select Infineon -> DAVE™ project (Figure 16). Click “Next”.

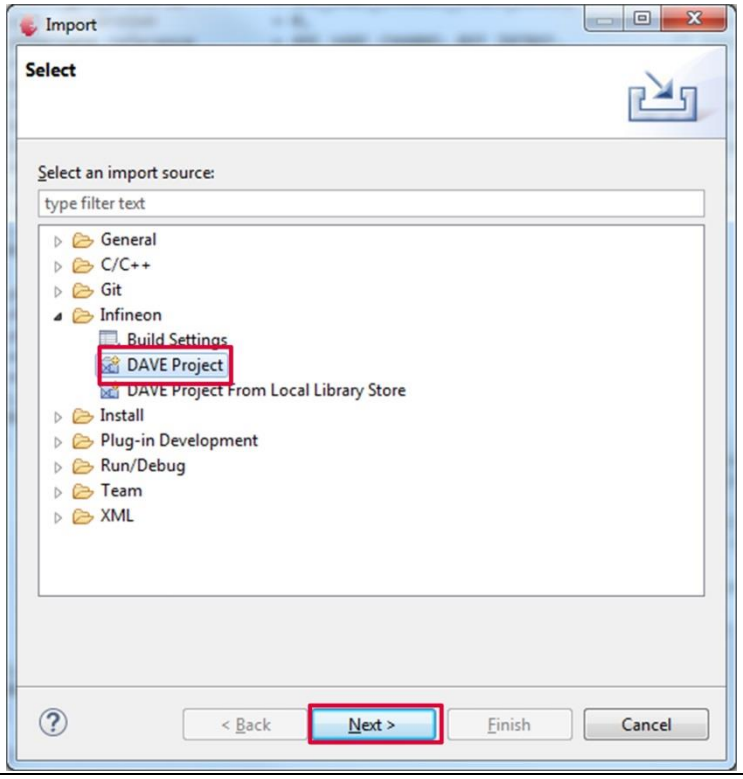

Figure 16 Select DAVE™ project

3. Select “Select Archive File”. Click “Browse”. Select the zipped project file from the location where it was previously downloaded to (Figure 17).

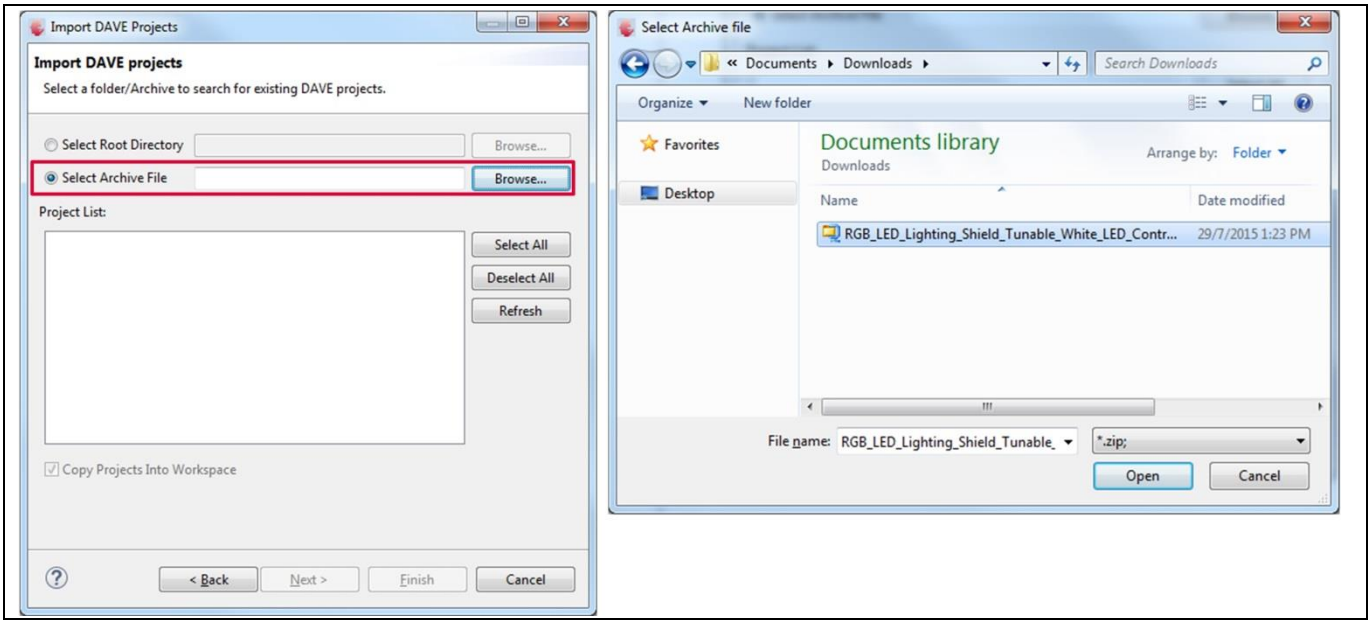

Figure 17 Select downloaded zipped file

4. The project name should appear in the project list (Figure 18). Click “Finish”.

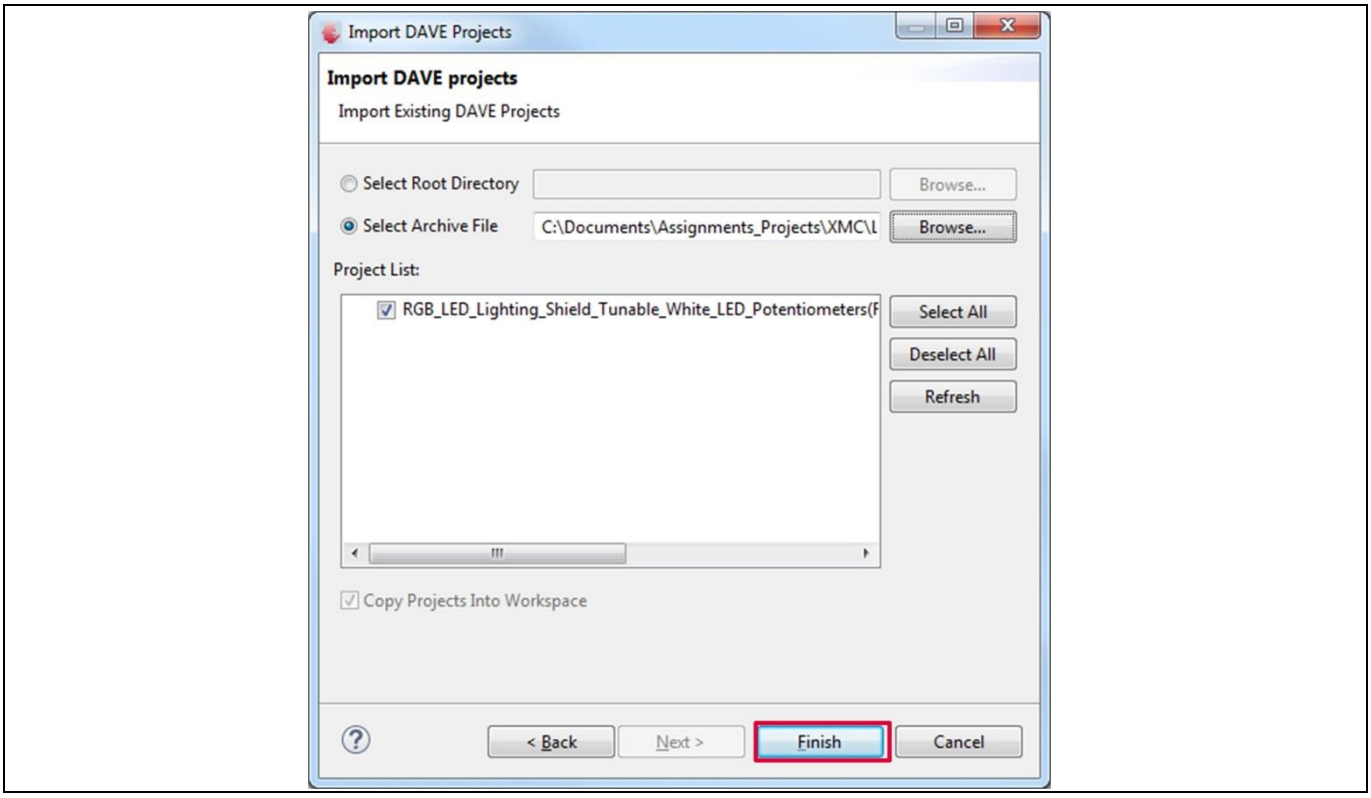

Figure 18 Project name appears under “Project list”

5. The project should appear in the DAVE™ workspace. Rebuild the project by clicking the “Rebuild active project” icon on the DAVE™ IDE toolbar (Figure 19).

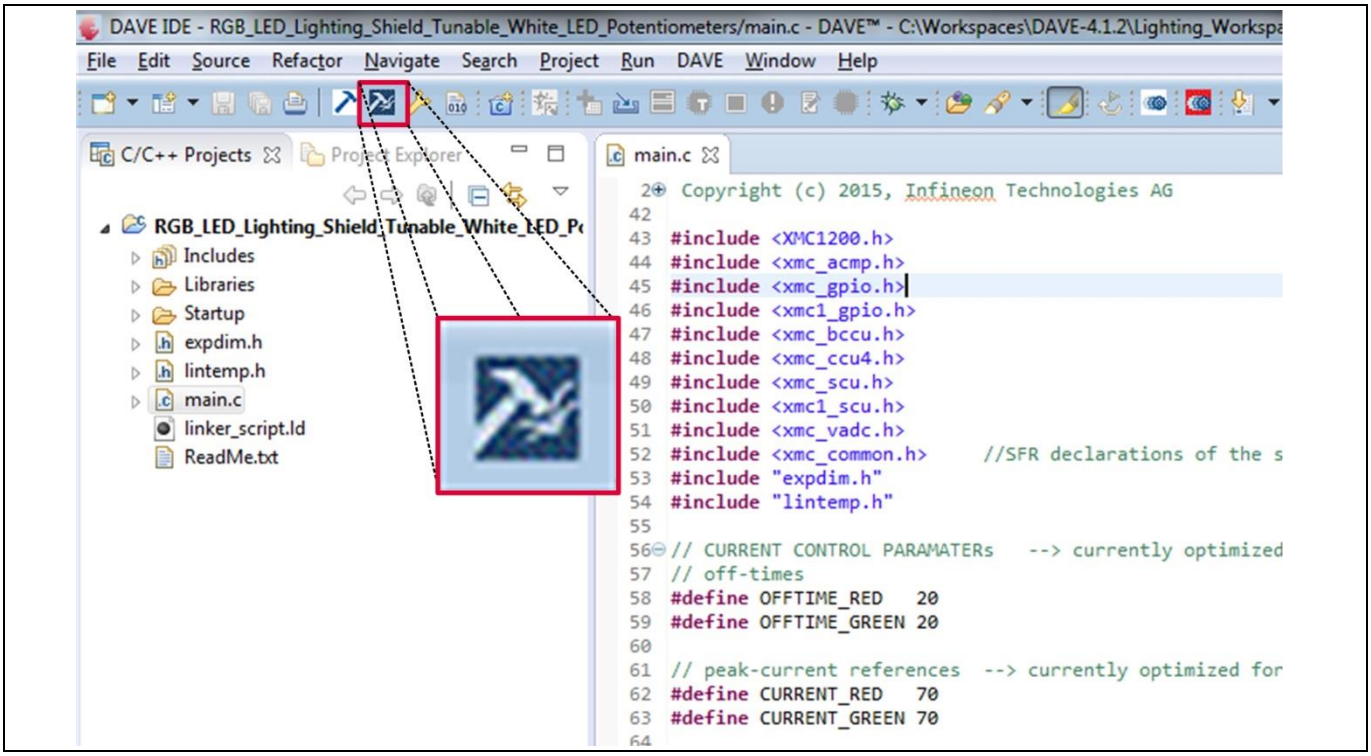

Figure 19 Rebuild the project

4.4 Flashing the XMC1202

To update the flash content of the onboard XMC1202 with the downloaded project:

1. Connect the boards together. The on-board XMC1202 microcontroller can be programmed over SWD via the debug interfaces using either a Segger J-Link LITE CortexM-9 or similar (Figure 20) or an XMC™ link debugger which will be available in September 2015.

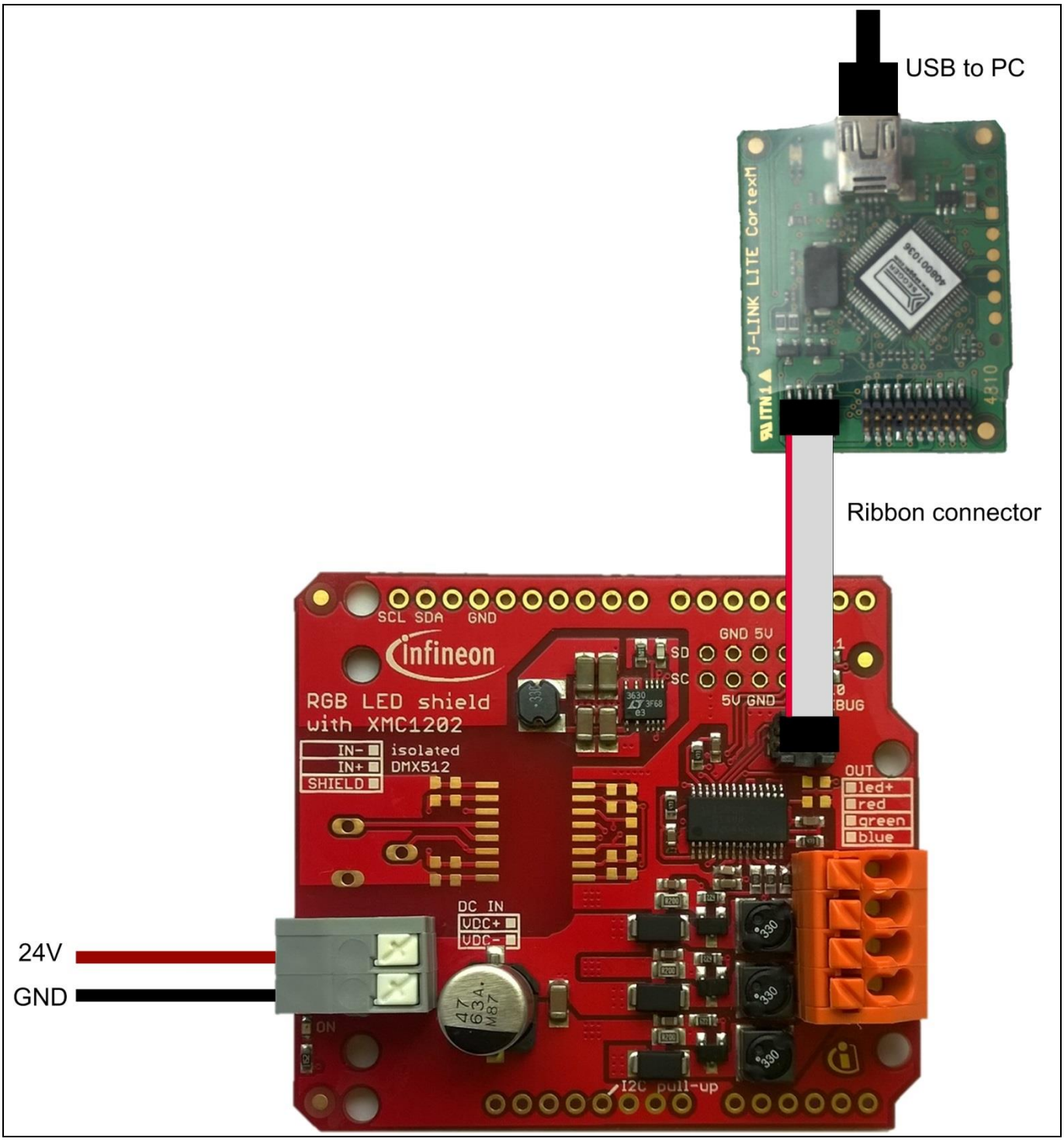

Figure 20 Segger J-Link LITE CortexM-9 connected to the RGB LED lighting shield

2. Click the debug icon on the DAVE™ IDE toolbar (Figure 21). This launches the “Debug configurations” window.

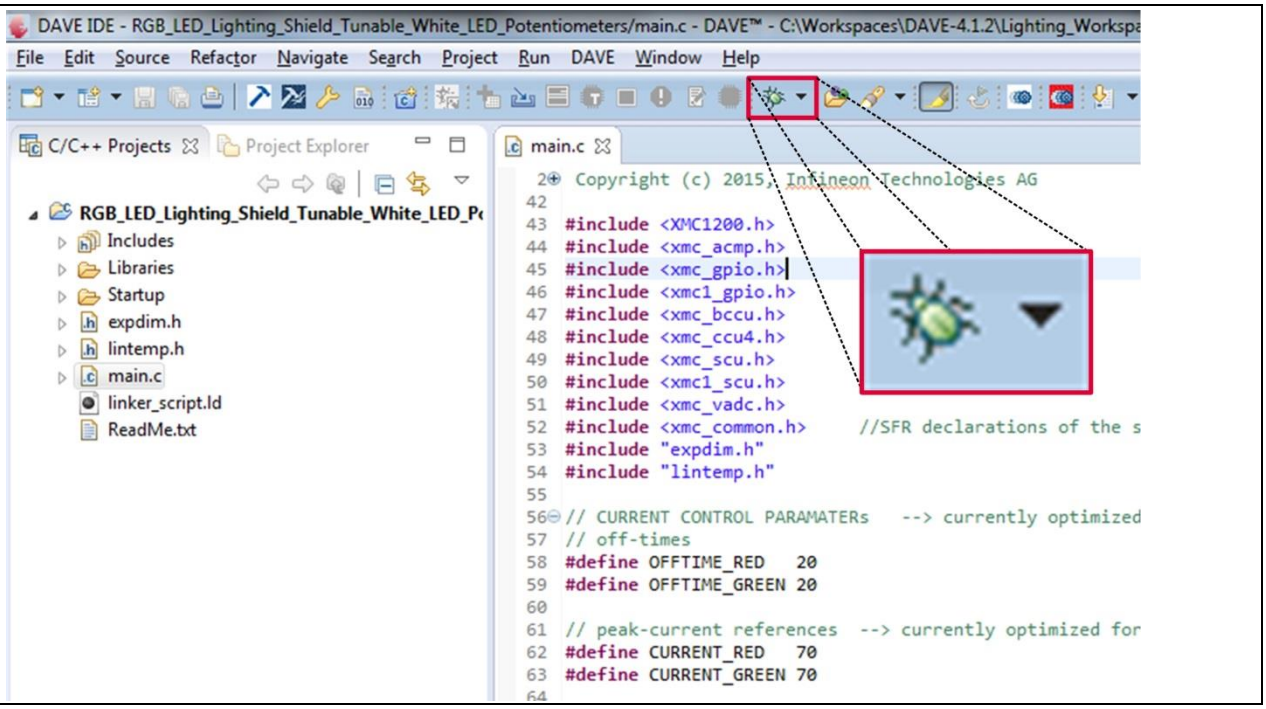

Figure 21 Debug icon

3. Double-click “GDB SEGGER J-Link debugging” (Figure 22). This creates a debug profile.

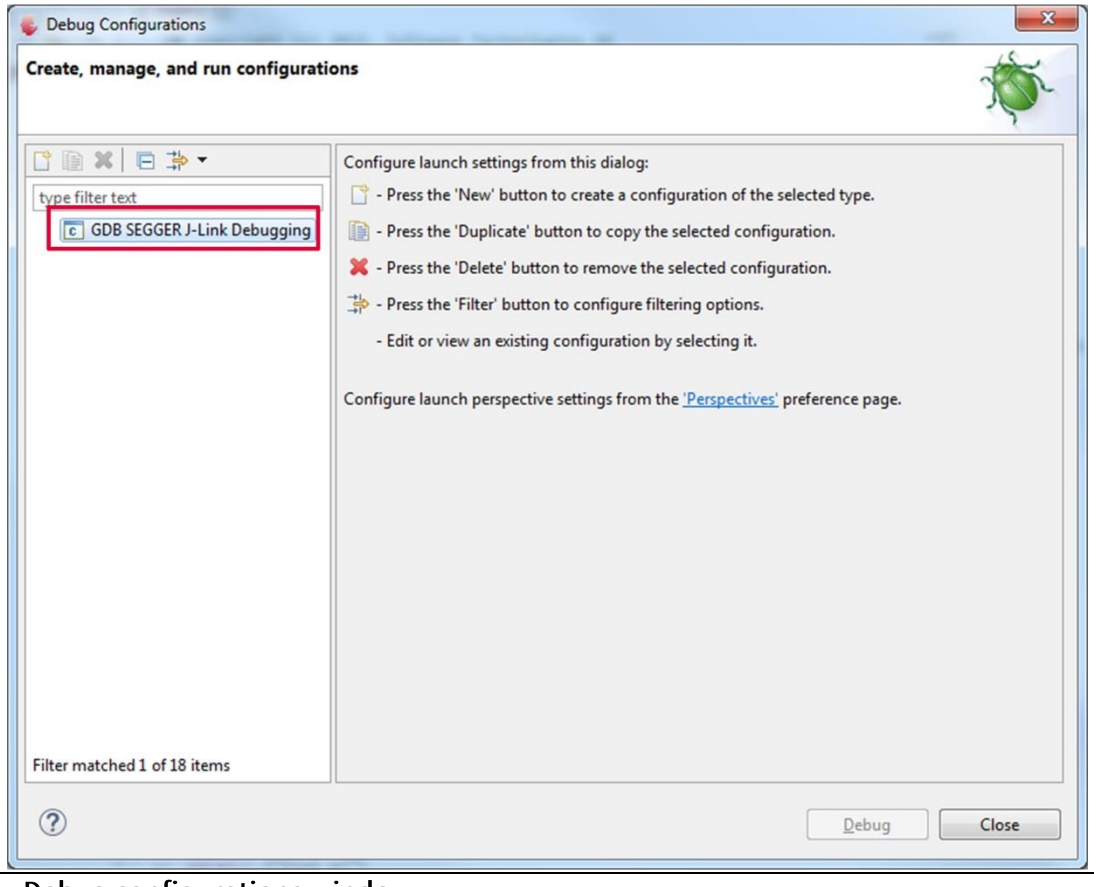

Figure 22 Debug configurations window

4. Open the “Debugger” tab (Figure 23). Ensure that “SWD” is selected as the debug interface. Click “Debug”.

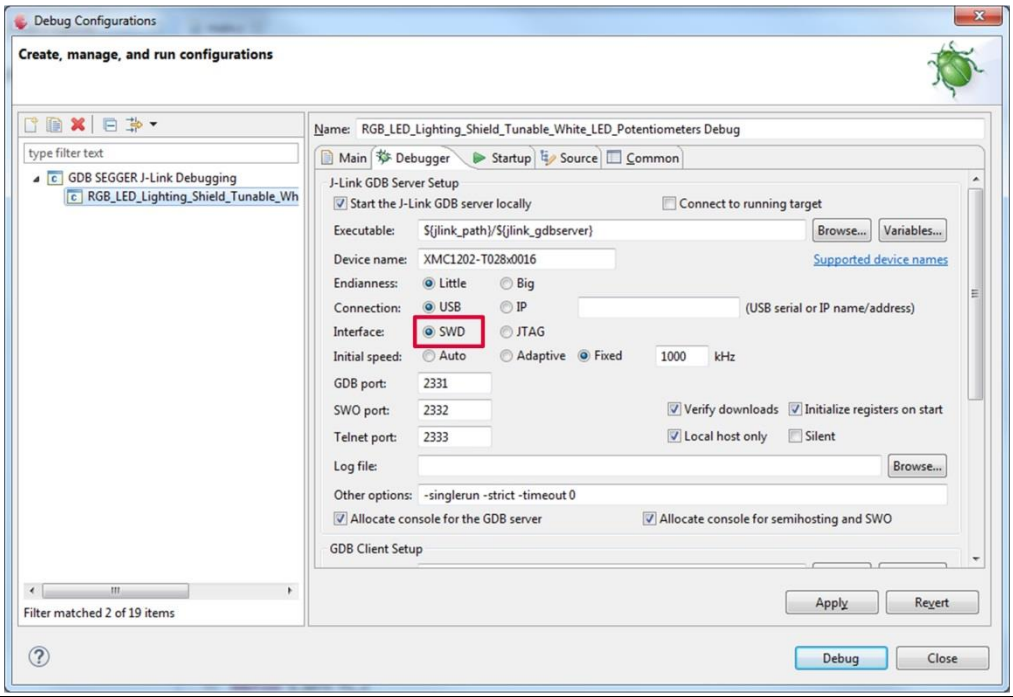

Figure 23 Debug profile configurations

5. The firmware is now programmed to the XMC1202 microcontroller (you can monitor the progress). Once finished, the debug perspective is launched (Figure 24).

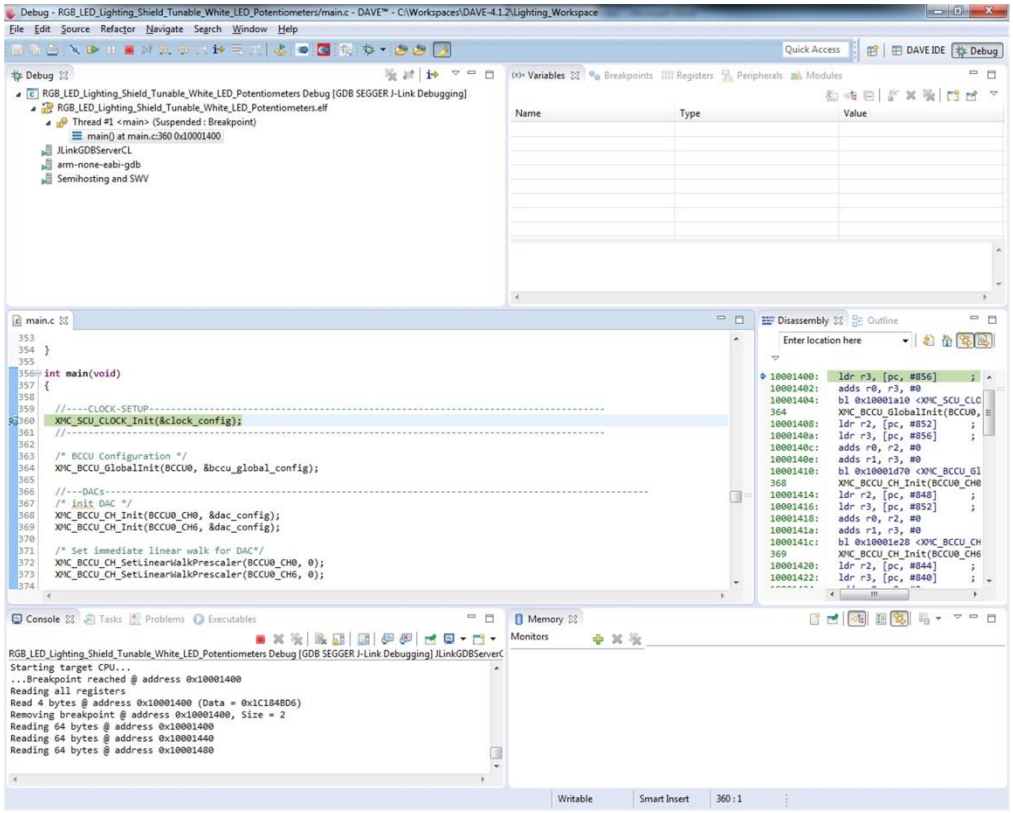

Figure 24 Debug perspective

6. Click Ctrl+F2 or the terminate icon on the DAVE™ IDE toolbar (Figure 25) to terminate debugging.

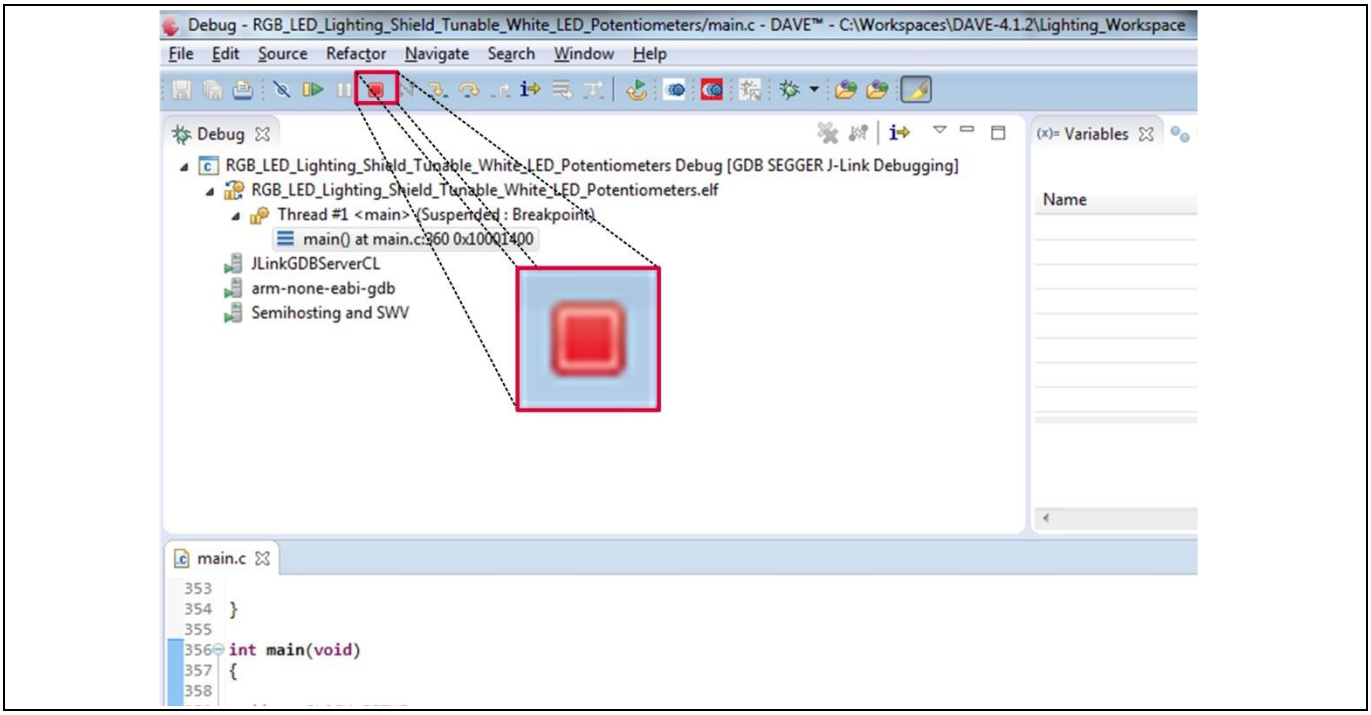

Figure 25 Terminate icon to terminate debug mode

4.5 Running the program

To start running the program:

1. Disconnect the debugger hardware from the RGB LED lighting shield.
2. Connect the tunable white LED lamp to the RGB LED lighting shield.
3. Connect the DC supply to the RGB LED lighting shield.
4. Turn the power ON.
5. Adjust the brightness level and color temperature of the lamp via the potentiometers.

5 Design considerations for better performance

In this section, at first, some background information is provided to explain the design decisions made for the RGB LED lighting shield. Then, some tips are offered to achieve a design with better performance than the one used for the RGB LED lighting shield.

5.1 Continuous Conduction Mode (CCM) buck LED driver design in the RGB LED lighting shield

The RGB LED lighting shield uses an XMC1202 microcontroller. A CCM buck LED driver design can be realized with an XMC1202 microcontroller and external circuitry using a combination of its peripherals. The control methodology or technique implemented for the CCM buck is called peak-current detection with fixed MOSFET off-time. This basically means that once the current rises to the desired peak level, the MOSFET is turned off for a fixed period of time to allow the current to drop. Then the MOSFET is turned on again and the current will start to rise again. This cycle is repeated over time. The result is to maintain an average current level. Figure 26 provides an illustration of this control technique.

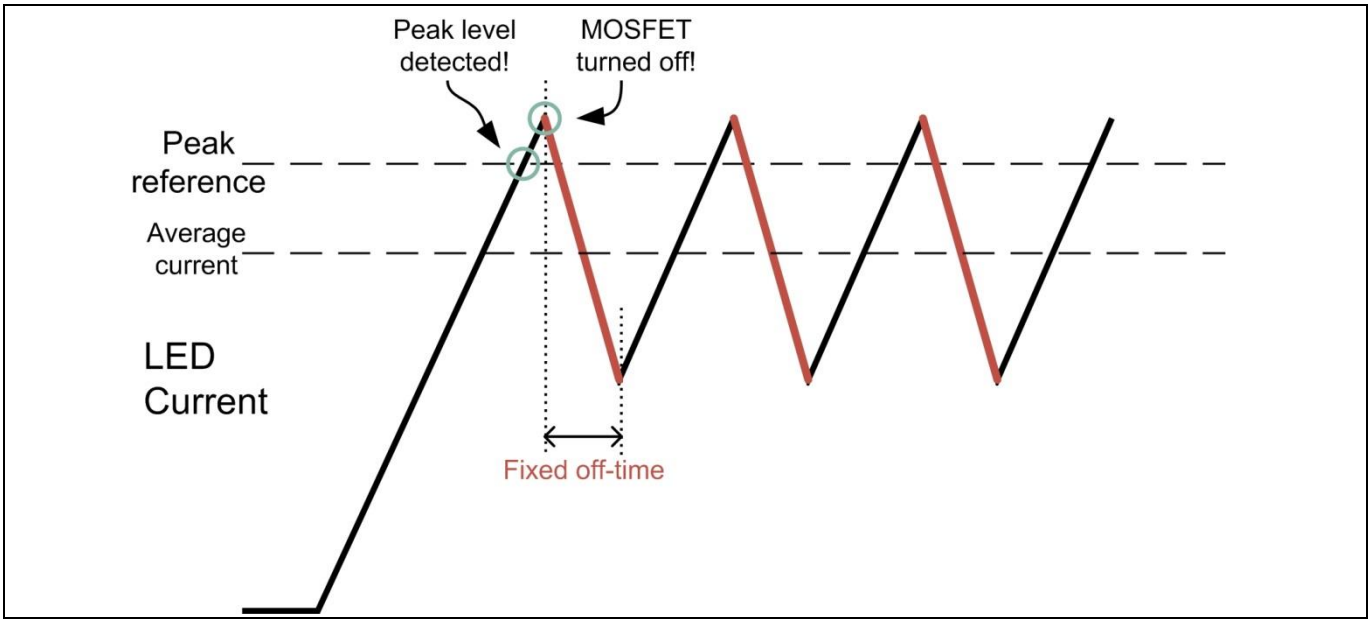

Figure 26 Peak-current detection with fixed MOSFET off-time

To realize this control technique, a peak level detector and a timer is required to work in tandem. With the XMC1202 microcontroller, an analog comparator slice (ACMP) can perform the role of the peak level detector whereas a Capture/Compare Unit 4 (CCU4) slice can serve as the timer (Figure 27).

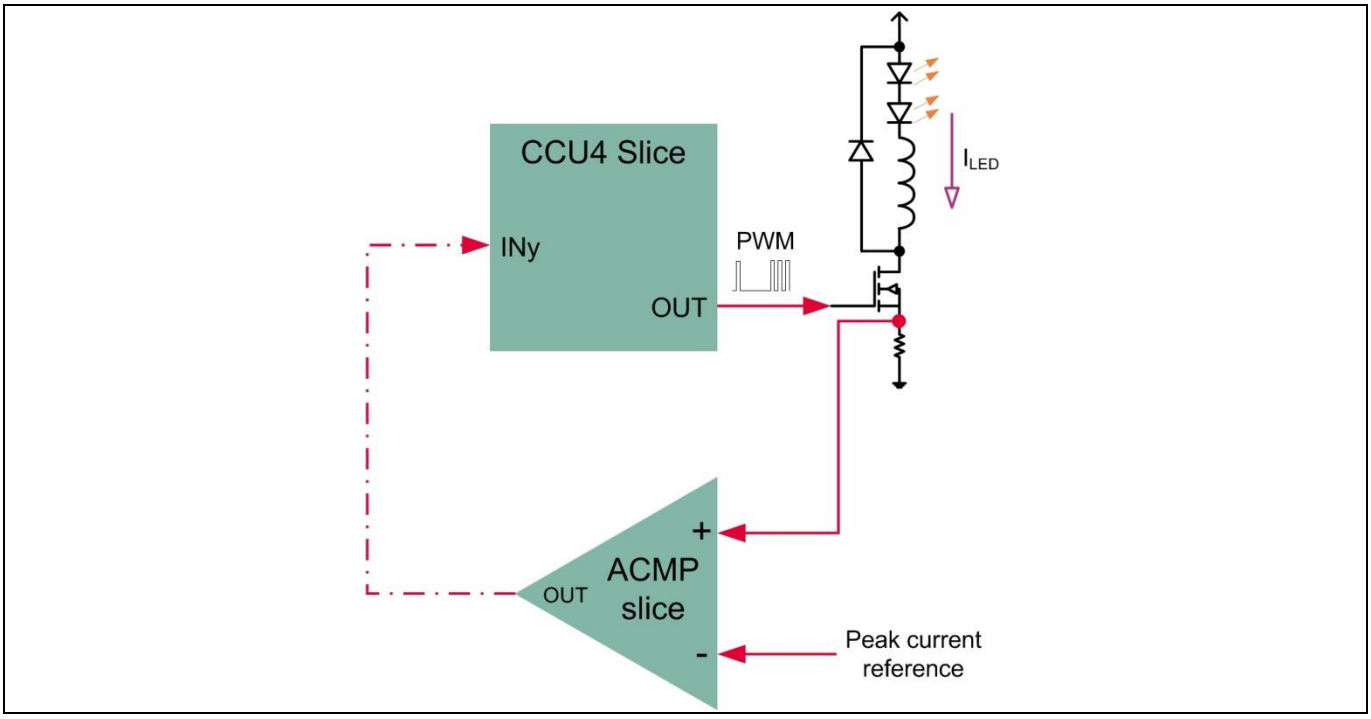

Figure 27 Implementation of CCM buck with XMC1000

The principle is for the analog comparator to trigger the CCU4 slice timer whenever the peak level is detected, so that timer can be reset, turning off the MOSFET. However, there is no direct interconnection between the ACMP and CCU4 slices in the XMC1202.

5.1.1 Routing options between ACMP and CCU4

Two options are available to route the output of the ACMP slice to the input of the CCU4 slice. The first option involves the Event Request Unit (ERU) to relay to ACMP output to the CCU4 slice (Figure 28).

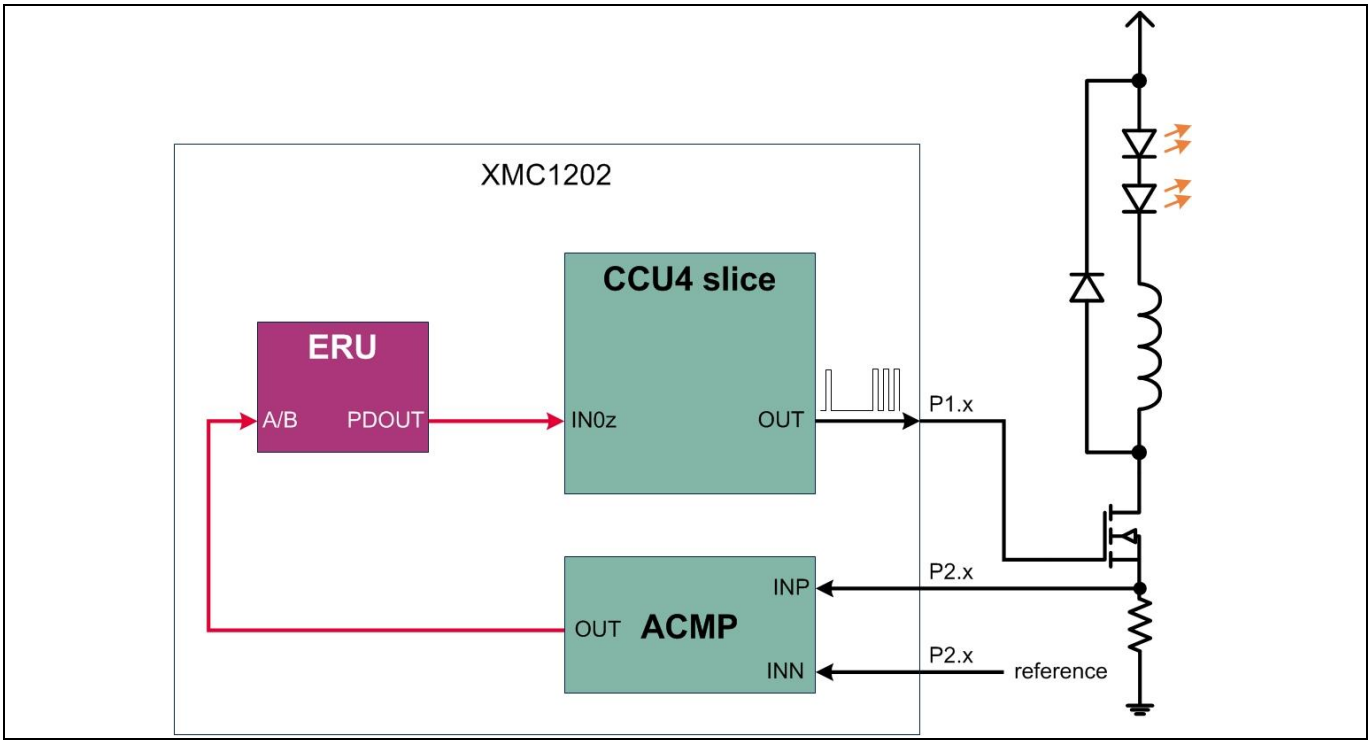

Figure 28 Routing ACMP to CCU4 via ERU

The second option involves routing both ACMP output and CCU4 input to physical microcontroller pins and then shorting these pins on the board (Figure 29).

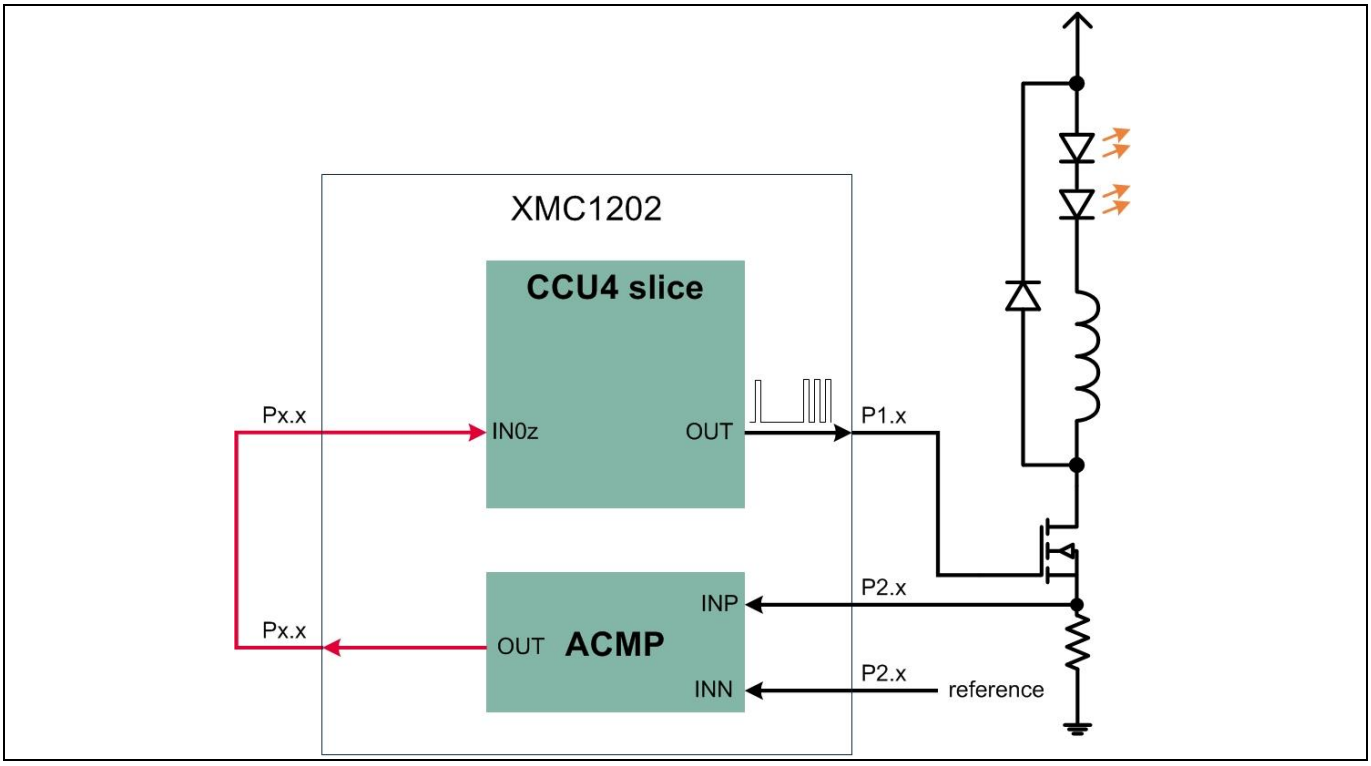

Figure 29 Routing ACMP to CCU4 via physical pins

5.1.2 Propagation delay in routes

The ACMP has an inherent delay between the detection at its input and signal triggering at its output. This delay is measured to be approximately 360 ns. The route between the ACMP output and CCU4 input adds further delay to the setup. This total delay from the point the detection is made at the ACMP input to the point the CCU4 timer is reset, is defined as the propagation delay. This results in an overshoot observed on the LED current (Figure 30).

The propagation delay in the two routing options mentioned above is as follows (Table 2):

Table 2 Propagation delay in routes

Route option	Propagation delay
ACMP-ERU-CCU4	440 ns (Figure 31)
ACMP-Pins-CCU4	360 ns (Figure 32)

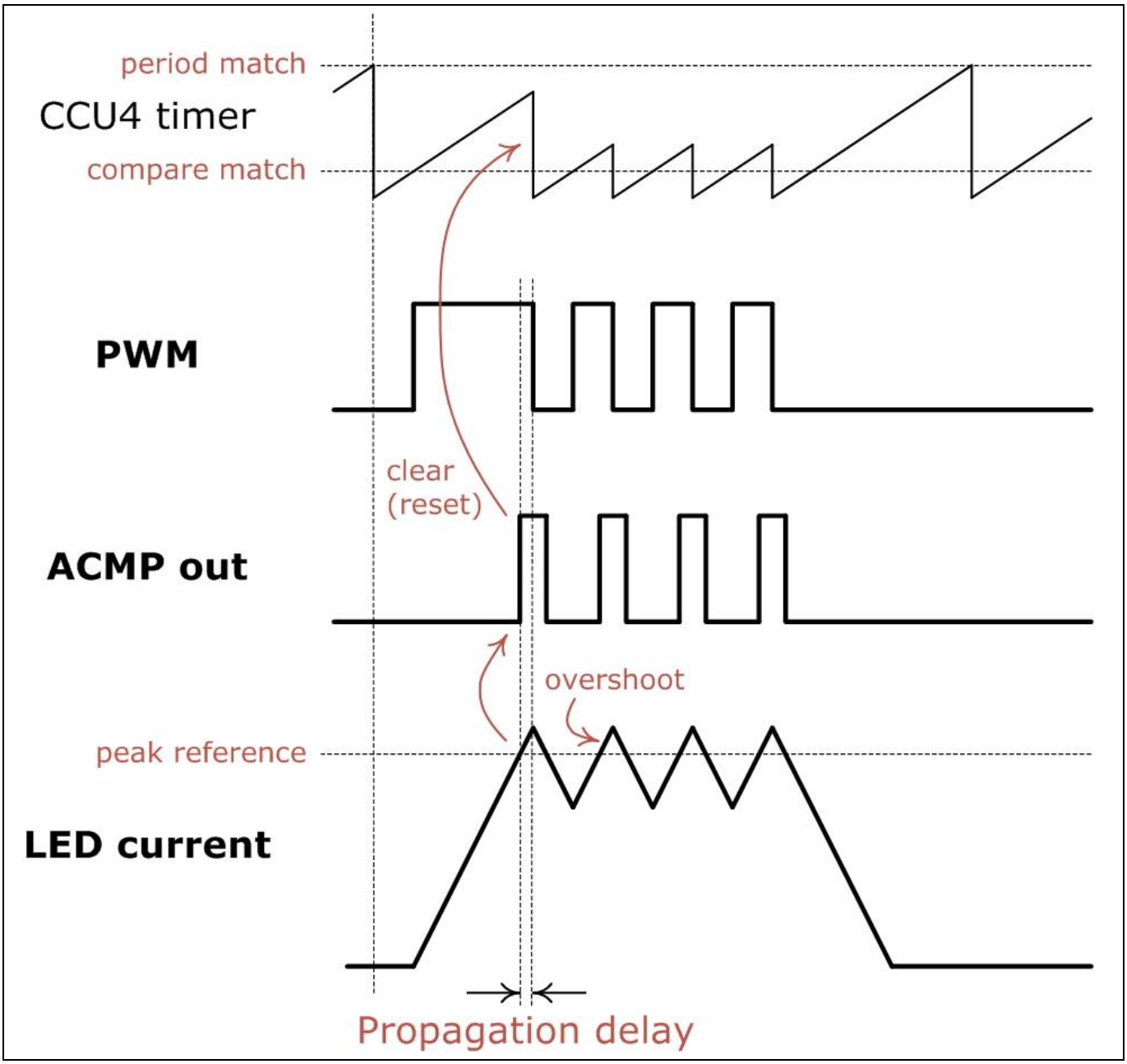

Figure 30 LED current overshoot due to propagation delay

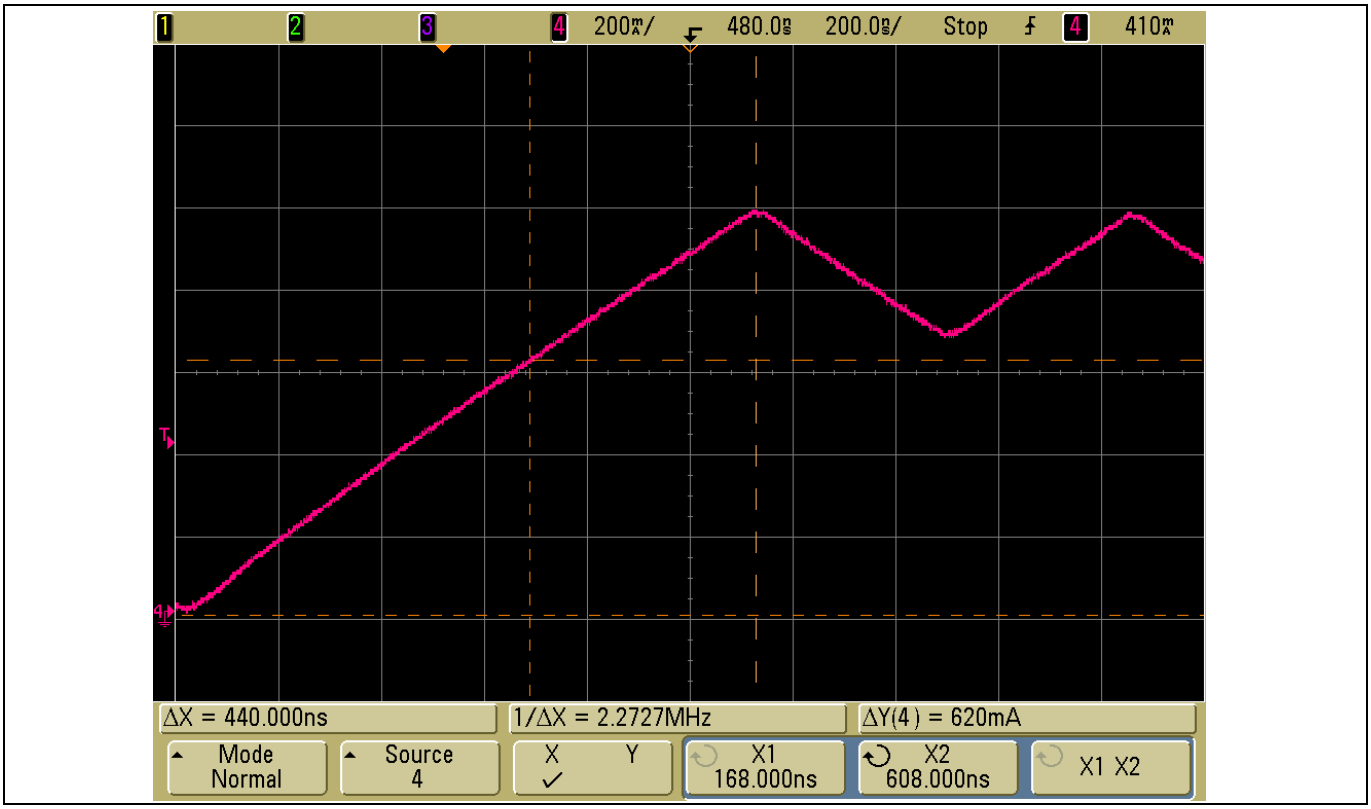

Figure 31 Propagation delay in route option 1

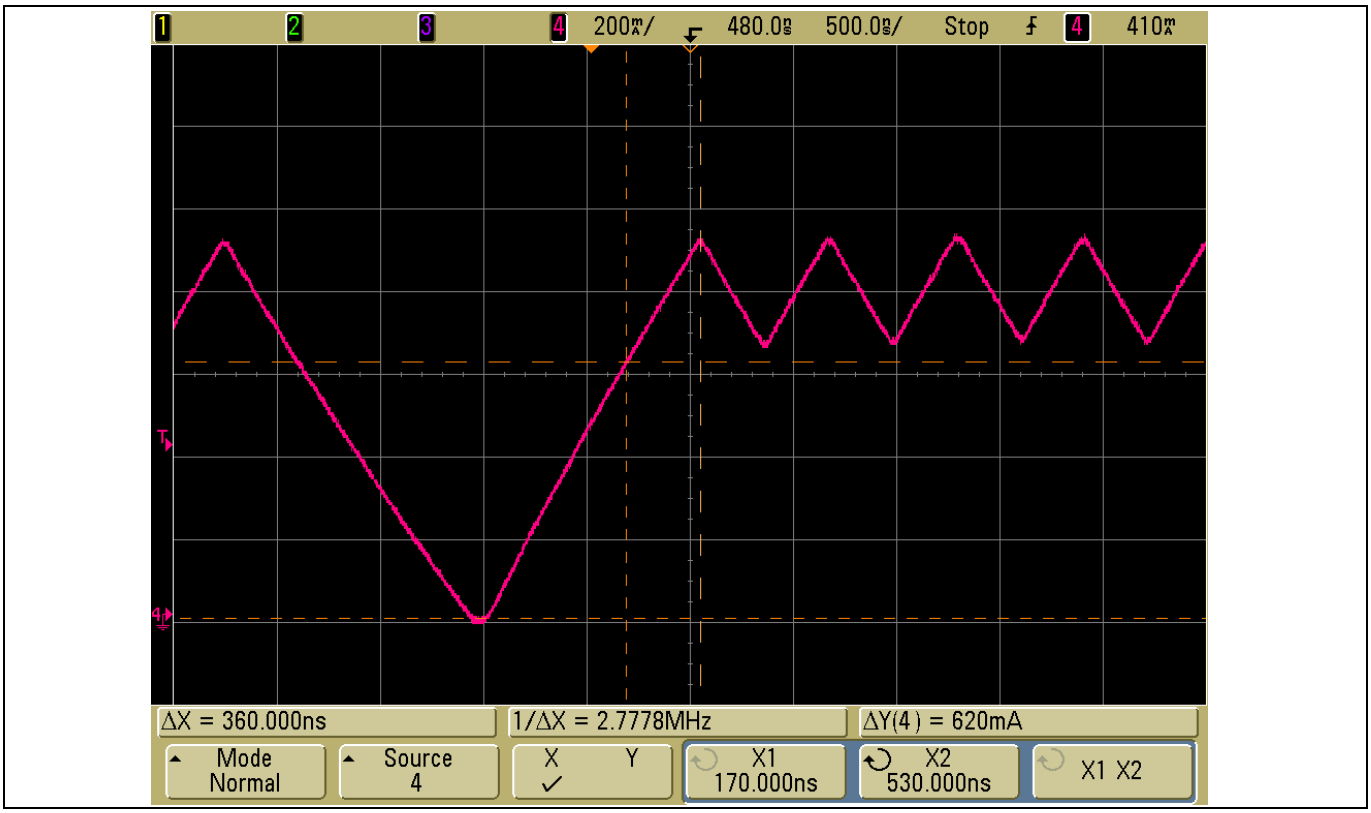

Figure 32 Propagation delay in route option 2

The RGB LED lighting shield uses option 2 (Figure 33) due to its shorter propagation display. However, this is at the expense of two physical pins per channel.

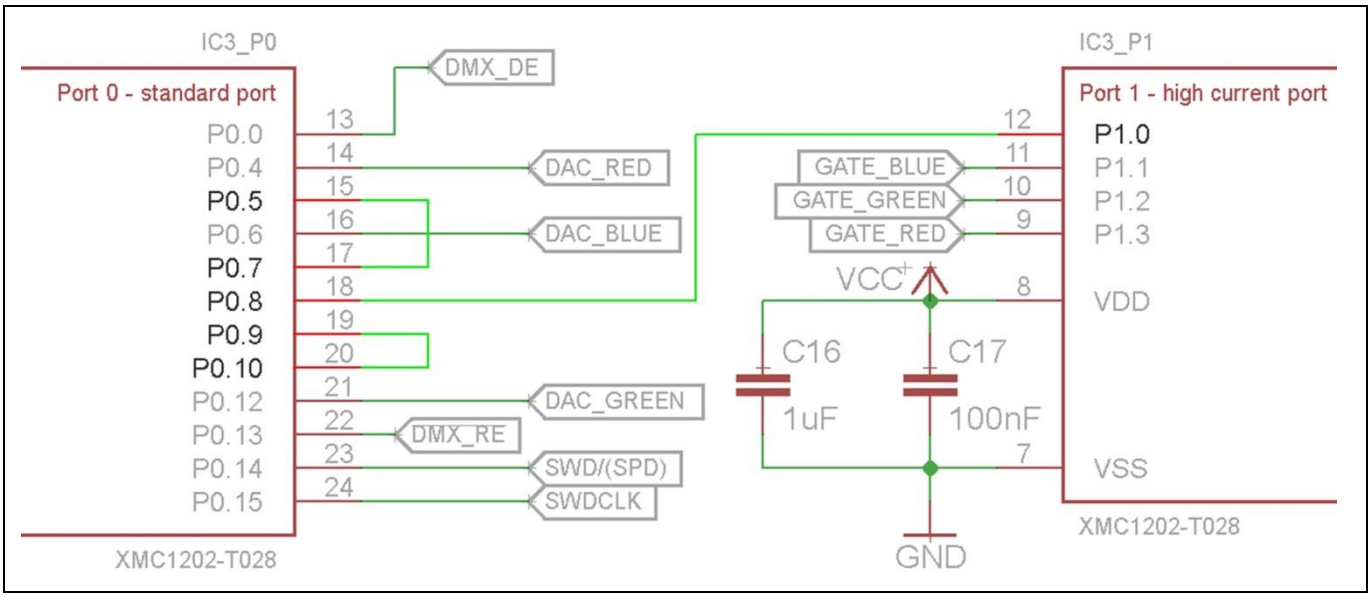

Figure 33 Shorted pins on RGB LED lighting shield

5.1.3 Dimming control for CCM buck LED driver with XMC120x

Dimming information is controlled via a Brightness and Color Control Unit (BCCU) channel. The BCCU channel's output, which is pulse-density modulated (PDM), is used to modulate the output of the CCU4 timer. The connection required is from the BCCU channel output to the CCU4 slice multi-channel input (MCI) (Figure 34).

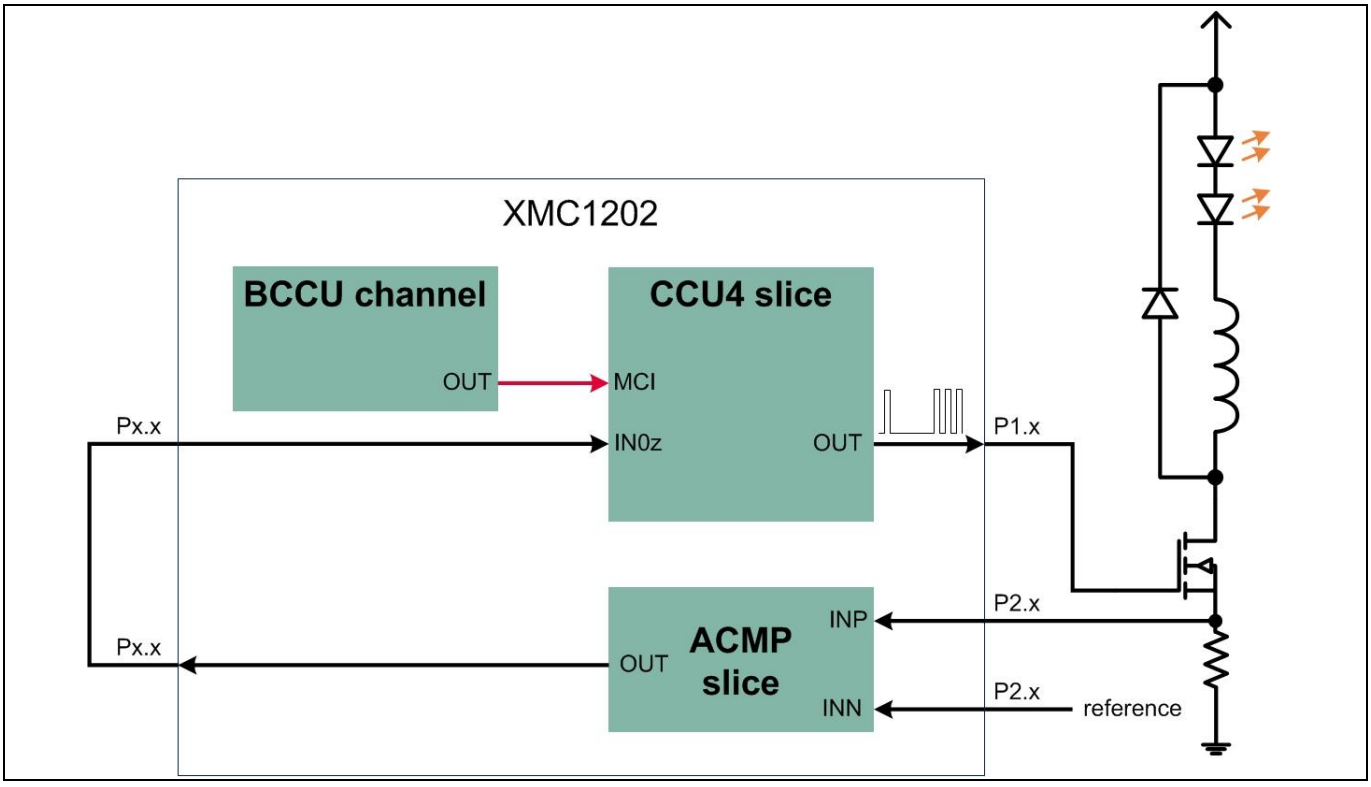

Figure 34 BCCU for dimming control

Figure 35 illustrates how the BCCU channel's PDM output is used to realize dimming control on the CCM buck LED driver. From this illustration, it can also be deduced that it is important for the propagration delay to be as small as possible, so that a higher frequency PDM signal can be used to modulate the PWM output. The advantage of using a higher frequency PDM is higher quality light.

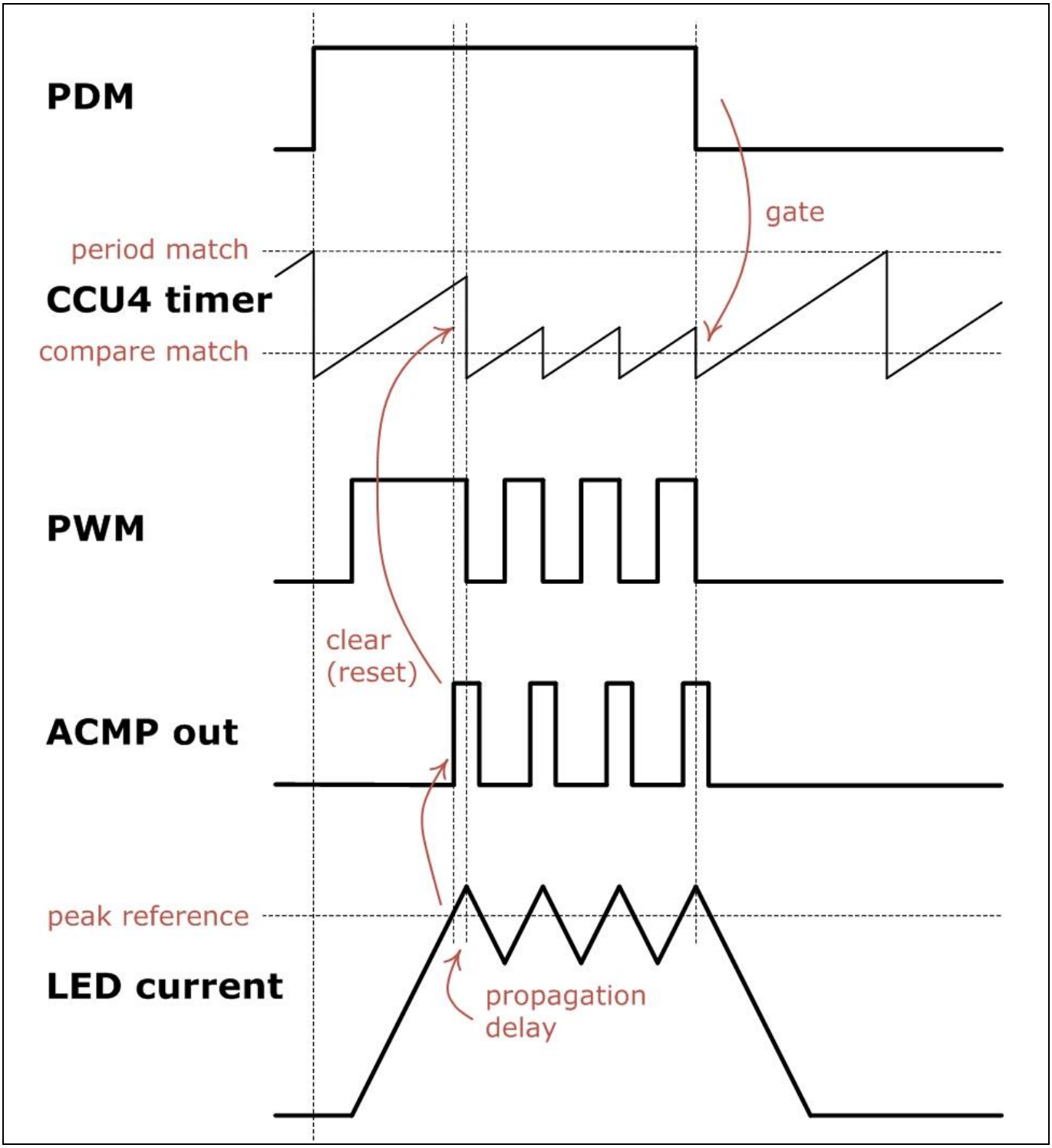

Figure 35 BCCU channel's PDM output gating CCU4 timer PWM output

5.2 Tips to improve design

The design of a CCM buck LED driver with XMC1000 microconcontrollers can be improved with a change of device to XMC130x or XMC140x. The peripherals and interconnect combination in the XMC130x and XMC140x allow for a shorter propagation delay and better dimmability.

5.2.1 Routing option in XMC130x and XMC140x

In the XMC130x and XMC140x, the ACMP slice output can be routed to the CCU8 slice input via a BCCU channel (Figure 36). This route is advantageous as a BCCU channel is always required for dimming control.

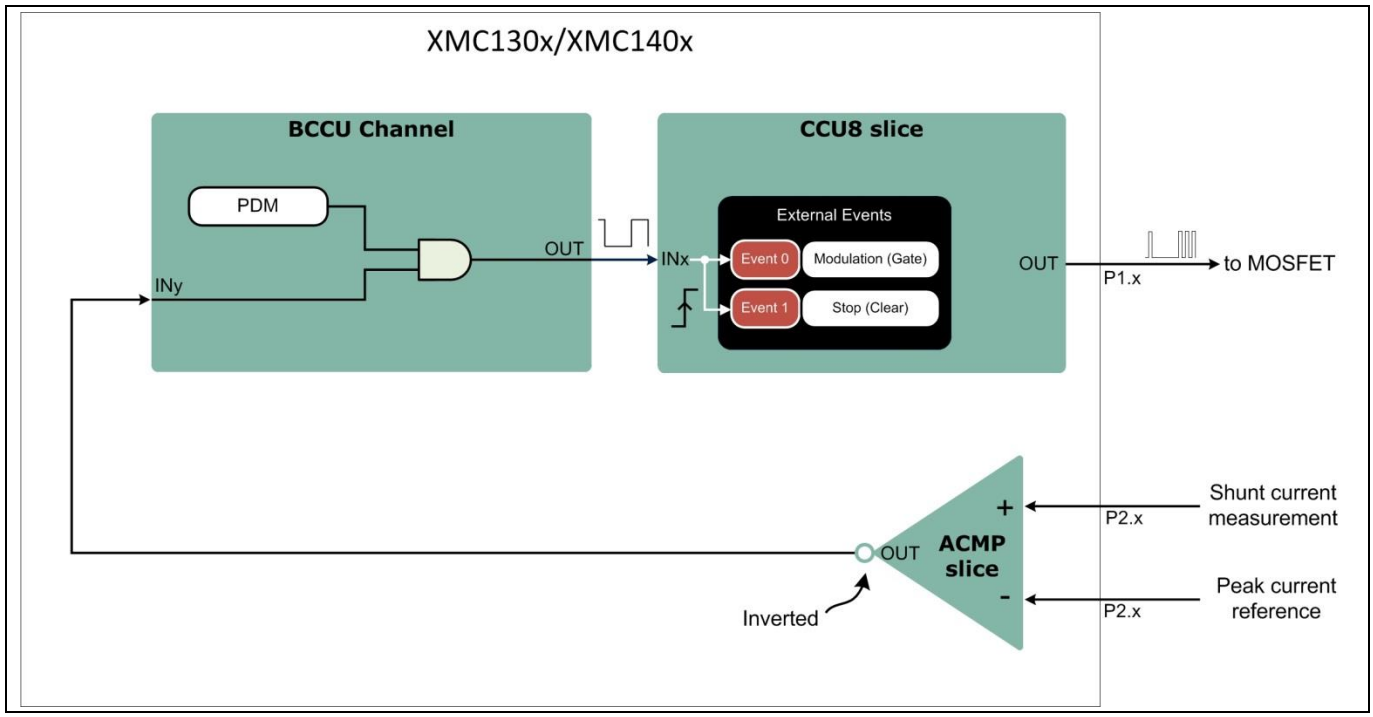

Figure 36 Routing ACMP to CCU8 via BCCU

This routing option also removes the need for additional physical pins for the connection.

5.2.2 Propagation delay with XMC130x and XMC140x

The propagation delay with this routing option is 272 ns, which is shorter than those measured in XMC1202 (Figure 37).

Figure 38 illustrates the LED current control signals with this routing option.

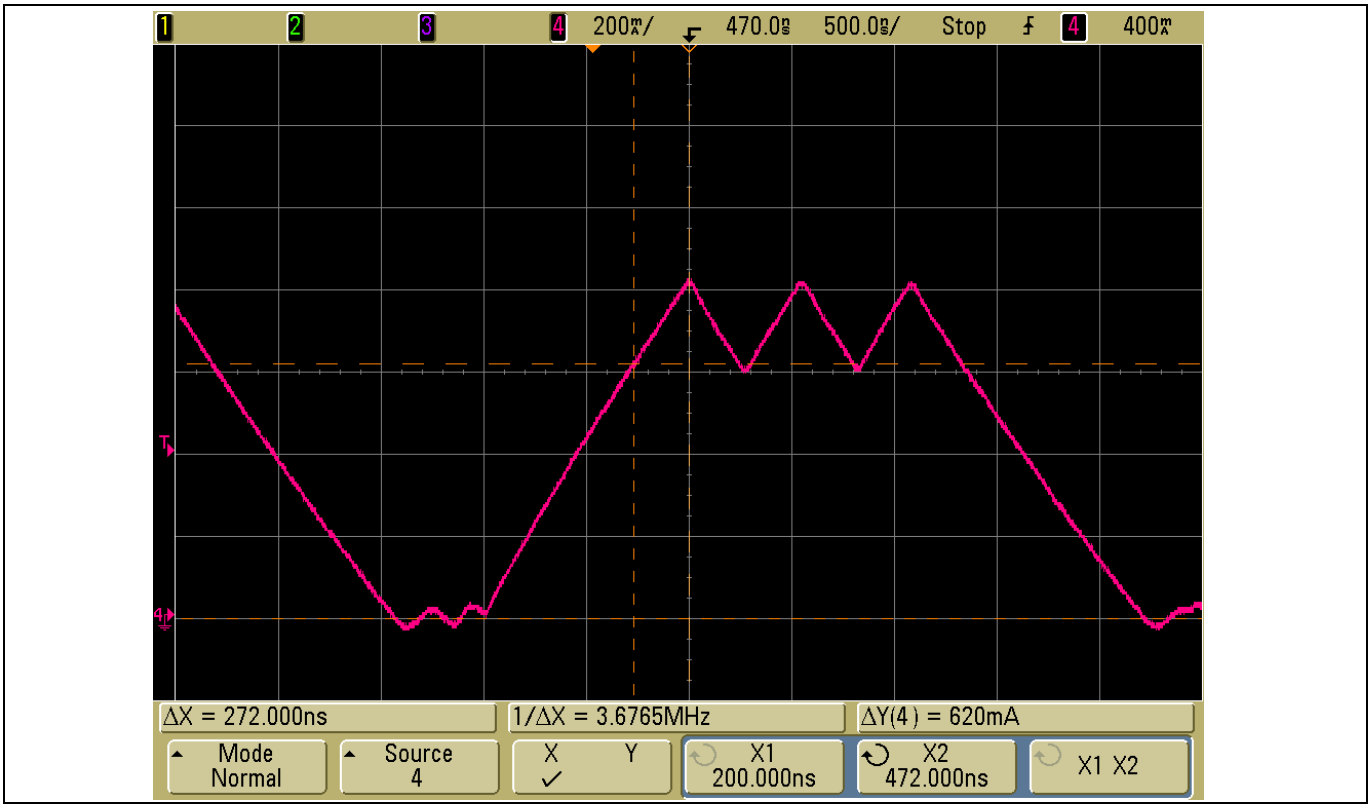

Figure 37 Propagation delay in XMC130x and XMC140x

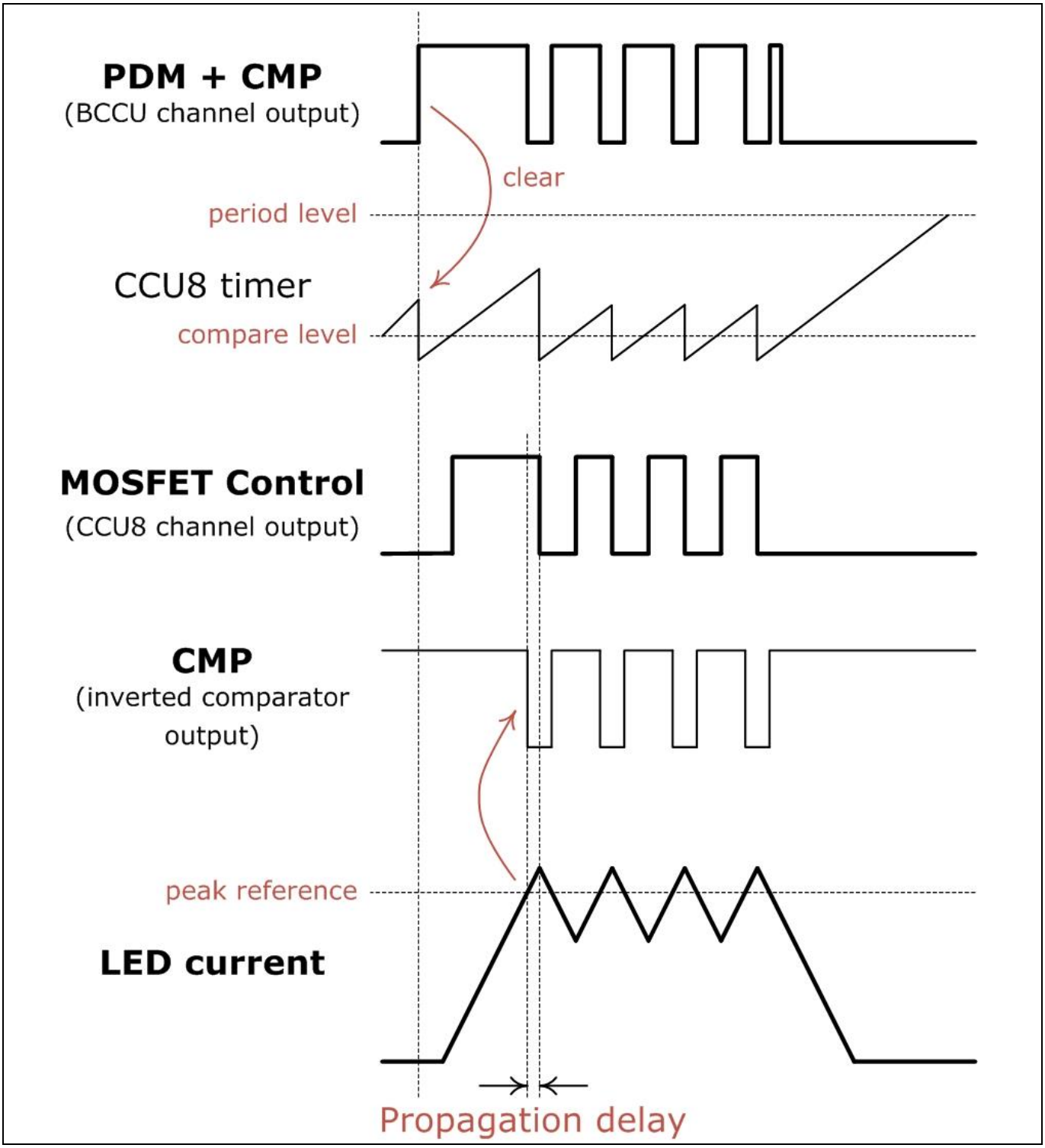

Figure 38 LED current control signals using ACMP-BCCU-CCU8 combination

5.2.3 Hardware migration

To migrate the design from the RGB LED lighting shield to one that uses an XMC1302 or XMC1402 for tunable white LED control, the following is proposed for minimal changes (Table 3):

Table 3 Resource reallocation for design with XMC1302 or XMC1402

	DAC (BCCU)	ACMP	BCCU	CCU8
CW Channel	P0.12 (Channel 6)	P2.2, P2.1 (Slice 2)	Channel 3	P1.2 (Slice 1)
WW Channel	P0.6 (Channel 2)	P2.6, P2.7 (Slice 1)	Channel 0	P1.1 (Slice 0)

Figure 39 provides an illustration of the resource reallocation as proposed above.

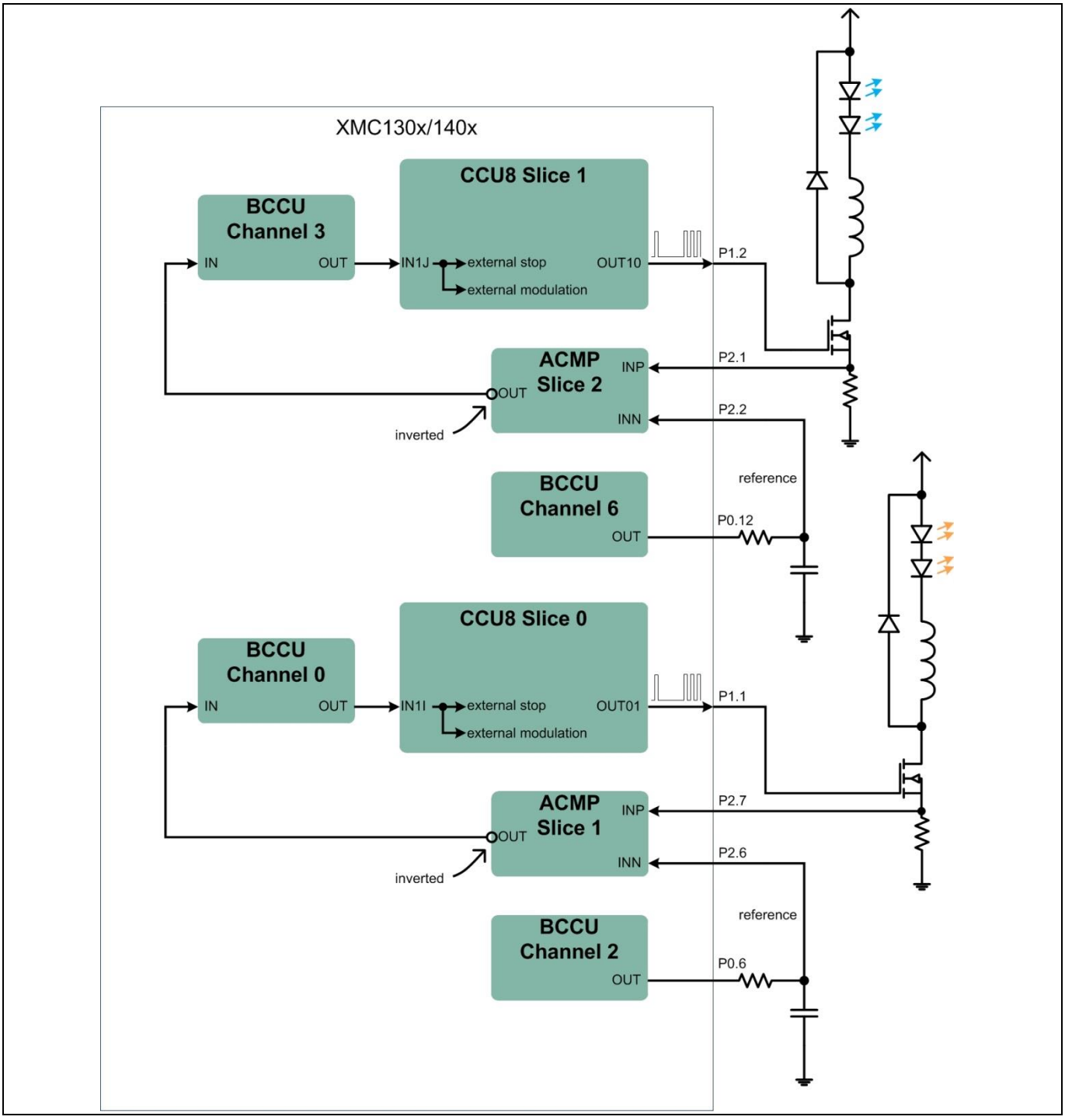

Figure 39 Resource reallocation for design with XMC1302 or XMC1402

5.2.4 Software adaptation

This section will provide the code to adapt the software for the proposed hardware changes and will be divided to two – Peripheral configuration data and Peripheral initialization.

These code lines can be copied into a new DAVE™ Easy Main project.

5.2.4.1 Peripheral configuration data

System clock

```
/* Clock configuration */
XMC_SCU_CLOCK_CONFIG_t clock_config =
{
  .pclk_src = XMC_SCU_CLOCK_PCLKSRC_DOUBLE_MCLK, /**< Source of PCLK is twice the MCLK. */
  .fdiv = 0, /**< Fractional clock divider not used */
  .idiv = 1 /**< Integer clock divider = 1 */
};
```

BCCU channel as pseudo-DAC for ACMP reference generation

```
/* 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 */
};
```

Analog comparator slice for peak-current detection

```
/* Configuration for ACMP slice used for peak-current detection */
XMC_ACMP_CONFIG_t cmp_config =
{
  .filter_disable = 0U, /**< Enable filter */
  .output_invert  = 1U, /**< Invert output */
  .hysteresis     = XMC_ACMP_HYSTERESIS_10 /**< Use 10mV hysteresis */
};
```

BCCU channel for dimming control

```
/* BCCU global configuration */
XMC_BCCU_GLOBAL_CONFIG_t bccu_global_config =
{
  .fclk_ps  = 0x50, /**< 800kHz @PCLK=64MHz */
  .dclk_ps  = 0x500, /**< 50kHz @PCLK=64MHz */
  .bclk_sel = XMC_BCCU_BCLK_MODE_NORMAL, /**< 200KHz @PCLK=64MHz */
  .maxzero_at_output = 204 /**< 0.5% min brightness */
};

/* BCCU trigger configuration */
XMC_BCCU_TRIG_CONFIG_t bccu_trig_config =
{
  .mode  = XMC_BCCU_TRIGMODE0, /**< trigger mode 0 */
  .delay = XMC_BCCU_TRIGDELAY_QUARTER_BIT, /**< BCCU trigger occurs on 1/4 bit time delayed
after channel trigger */
  .mask_chans = 0x09 /**< channel 3 */
};
```

```
/* Configuration for BCCU channel used as PDM (pulse-density modulator) for CW channel */
XMC_BCCU_CH_CONFIG_t cw_pdm_config =
{
  .pack_en         = 1, /**< enable packer */
  .force_trig_en   = 0, /**< disable forced trigger */
  .trig_edge       = XMC_BCCU_CH_TRIG_EDGE_ACT_TO_PASS, /**< trigger on falling 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 */
  .pack_offcnt_val = 0, /**< initial packer off count value */
  .pack_offcmp_lev = 128, /**< initial packer off compare value */
  .pack_oncmp_lev  = 3 /**< initial packer on compare value */
};

/* Configuration for BCCU channel used as PDM (pulse-density modulator) for WW channel */
XMC_BCCU_CH_CONFIG_t ww_pdm_config =
{
  .pack_en         = 1, /**< enable packer */
  .force_trig_en   = 0, /**< disable forced trigger */
  .trig_edge       = XMC_BCCU_CH_TRIG_EDGE_ACT_TO_PASS, /**< trigger on falling 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 */
  .pack_offcnt_val = 66, /**< initial packer off count value */
  .pack_offcmp_lev = 128, /**< 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 /**< immediate dimming */
};
```

CCU8 timer slice as MOSFET gate driver

```
/* 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 (CW Channel) */
XMC_CCU8_SLICE_EVENT_CONFIG_t cw_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 (CW Channel) */
XMC_CCU8_SLICE_EVENT_CONFIG_t cw_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 */
};

/* Configuration for CCU8 slice external event 0 (WW Channel) */
XMC_CCU8_SLICE_EVENT_CONFIG_t ww_slice_clear_event_config =
{
  .mapped_input = CCU80_IN0_BCCU0_OUT0, /**< 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 (WW Channel) */
XMC_CCU8_SLICE_EVENT_CONFIG_t ww_slice_mod_event_config =
{
  .mapped_input     = CCU80_IN0_BCCU0_OUT0, /**< 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 */

};
```

5.2.4.2 Peripheral initialization

System clock

```
/* Clock initialization */
XMC_SCU_CLOCK_Init(&clock_config);
```

BCCU channel as pseudo-DAC for ACMP reference generation

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

/* init BCCU channels to function as DACs */
XMC_BCCU_CH_Init(BCCU0_CH6, &dac_config);
XMC_BCCU_CH_Init(BCCU0_CH2, &dac_config);

/* Set immediate linear walk for DAC*/
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH6, 0);
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH2, 0);

/* Enable DAC channels */
XMC_BCCU_ConcurrentEnableChannels(BCCU0, 0x044);
```

```
/* set current reference to low now */
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH6, 70);
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH2, 70);

/* start linear walks so to effect reference level at output */
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x044);

/* initialize output pins for DACs */
XMC_GPIO_SetMode(P0_12, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT1);
XMC_GPIO_SetMode(P0_6, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT1);
```

Analog comparator slice for peak-current detection

```
/* Comparator initialization for CW channel */
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);

/* Comparator initialization for WW channel */
XMC_ACMP_Init(XMC_ACMP0, 1, &cmp_config);
XMC_ACMP_EnableComparator(XMC_ACMP0, 1);
XMC_ACMP_SetInput(XMC_ACMP0, 1, XMC_ACMP_INP_SOURCE_STANDARD_PORT);
```

BCCU channel for dimming control

```
/* 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 channels */
XMC_BCCU_CH_Init(BCCU0_CH3, &cw_pdm_config); /* for CW */
XMC_BCCU_CH_Init(BCCU0_CH0, &ww_pdm_config); /* for WW */

/* Set fast linear walk for PDM channels*/
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH3, 48); /* for CW */
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH0, 48); /* for WW */

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

/* Enable PDM channels */
XMC_BCCU_ConcurrentEnableChannels(BCCU0, 0x009);

/* go to initial dimming level */
XMC_BCCU_DIM_SetTargetDimmingLevel(BCCU0_DE0, 1024); /* 25% brightness */
XMC_BCCU_StartDimming(BCCU0, 0);

/* go to initial intensity levels (color temperatures) */
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH3, 2048); /* 50% warm */
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 2048); /* 50% cool */
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x009);
```

CCU8 timer slice as 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 slices out of idle mode */
XMC_CCU8_EnableClock(CCU80, 1); /* Timer slice for CW */
XMC_CCU8_EnableClock(CCU80, 0); /* Timer slice for WW */

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

/* Initialize the slices */
XMC_CCU8_SLICE_CompareInit(CCU80_CC81, &ccu8_mosfet_gate_config);
XMC_CCU8_SLICE_CompareInit(CCU80_CC80, &ccu8_mosfet_gate_config);

/* Program period  and compare values */
XMC_CCU8_SLICE_SetTimerPeriodMatch(CCU80_CC81, 0x27FU); /* Program period value for CW */
XMC_CCU8_SLICE_SetTimerCompareMatch(CCU80_CC81, XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 20); /*
Program compare value for CW */
XMC_CCU8_SLICE_SetTimerPeriodMatch(CCU80_CC80, 0x27FU); /* Program period value for WW */
XMC_CCU8_SLICE_SetTimerCompareMatch(CCU80_CC80, XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 20); /*
Program compare value for WW */

/* Enable shadow transfer */
XMC_CCU8_EnableShadowTransfer(CCU80, XMC_CCU8_SHADOW_TRANSFER_SLICE_1); /* Shadow transfer
for CW */
XMC_CCU8_EnableShadowTransfer(CCU80, XMC_CCU8_SHADOW_TRANSFER_SLICE_0); /* Shadow transfer
for WW */

/* External stop event configuration (CW) */
/* init event 0 as external stop event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC81, XMC_CCU8_SLICE_EVENT_0,
&cw_slice_clear_event_config);
/* configure external stop event to clear timer */
XMC_CCU8_SLICE_StopConfig(CCU80_CC81, XMC_CCU8_SLICE_EVENT_0,
XMC_CCU8_SLICE_END_MODE_TIMER_CLEAR);

/* External modulation event configuration (CW) */
/* init event 1 as external modulation event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC81, XMC_CCU8_SLICE_EVENT_1,
&cw_slice_mod_event_config);
/* configure external modulation event to gate output without synchronization */
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);

/* External stop event configuration (WW) */
/* init event 0 as external stop event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC80, XMC_CCU8_SLICE_EVENT_0,
&ww_slice_clear_event_config);
/* configure external stop event to clear timer */
XMC_CCU8_SLICE_StopConfig(CCU80_CC80, XMC_CCU8_SLICE_EVENT_0,
XMC_CCU8_SLICE_END_MODE_TIMER_CLEAR);

/* External modulation event configuration (WW) */
/* init event 1 as external modulation event */
XMC_CCU8_SLICE_ConfigureEvent(CCU80_CC80, XMC_CCU8_SLICE_EVENT_1,
&ww_slice_mod_event_config);
/* configure external modulation event to gate output without synchronization */
XMC_CCU8_SLICE_ModulationConfig(CCU80_CC80, 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);
XMC_CCU8_SLICE_StartTimer(CCU80_CC80);

/* enable output pins */
XMC_GPIO_SetMode(P1_2, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5);
XMC_GPIO_SetMode(P1_1, XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT5);
```

Revision history

Major changes since the last revision

Page or reference	Description of change
5 Design considerations for better performance	Chapter explaining design in RGB LED lighting shield and a guide to optimizing design.

Edition 2016-09-15

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AP32314

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.