Pseudo Digital-to-Analog Converter (DAC) with XMC1000

XMC1000

About this document

Scope and purpose

This document provides guidance for using switching peripherals such as the Brightness and Color Control Unit (BCCU), Capture/Compare Unit 4 (CCU4) and Capture/Compare Unit 8 (CCU8) in the XMC1000 Microcontroller as pseudo Digital-to-Analog Converters (DACs) in the absence of built-in DACs.

Intended audience

This document is intended for engineers who wish to implement DAC functionality in their application with the XMC1000 Microcontroller.

Applicable products

- XMC110x
- XMC120x
- XMC130x
- XMC140x
- DAVE ™

References

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

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

Table of contents

1 Introduction

Many embedded microcontroller applications require the generation of analog signals. Sometimes, a dedicated DAC IC is used for this purpose. Some microcontrollers, like the XMC4000 Microcontroller family, have a built-in DAC peripheral, eliminating the need for an external IC. In the absence of a DAC peripheral such as XMC1000 Microcontroller family, a modulation peripheral such as the BCCU, CCU4 and CCU8 can be used to create a pseudo Digital-to-Analog Converter (DAC) with the help of discrete low pass filter.

This application note provides hints on how the user can implement the DAC functionality, illustrated with some example use cases.

1.1 Theory of operation

The basic principle of a pseudo DAC can be illustrated with a simple Pulse Width Modulation (PWM) signal. The duty cycle (ON time) is proportional to the average value of the peak signal.

When the PWM signal is passed through a low pass filter with low cut-off frequency, it removes high frequency components and leaves the average value at its output. This behavior is illustrated in Figure 1.

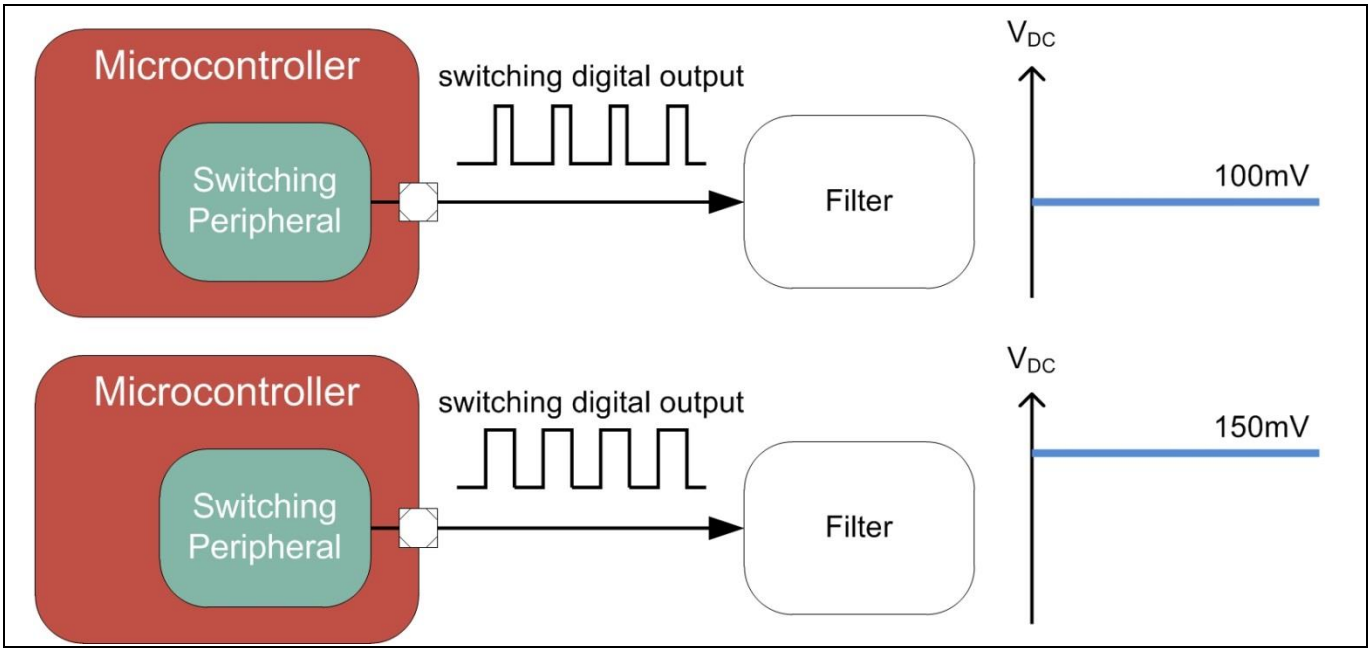

Figure 1 Pseudo DAC theory of application with PWM signal

A pseudo DAC can also be implemented with Pulse Density Modulation (PDM). When the PDM signal is passed through a low pass filter with low cut-off frequency, the average value of such signal is proportional to the pulse density. Such behavior is illustrated in Figure 2.

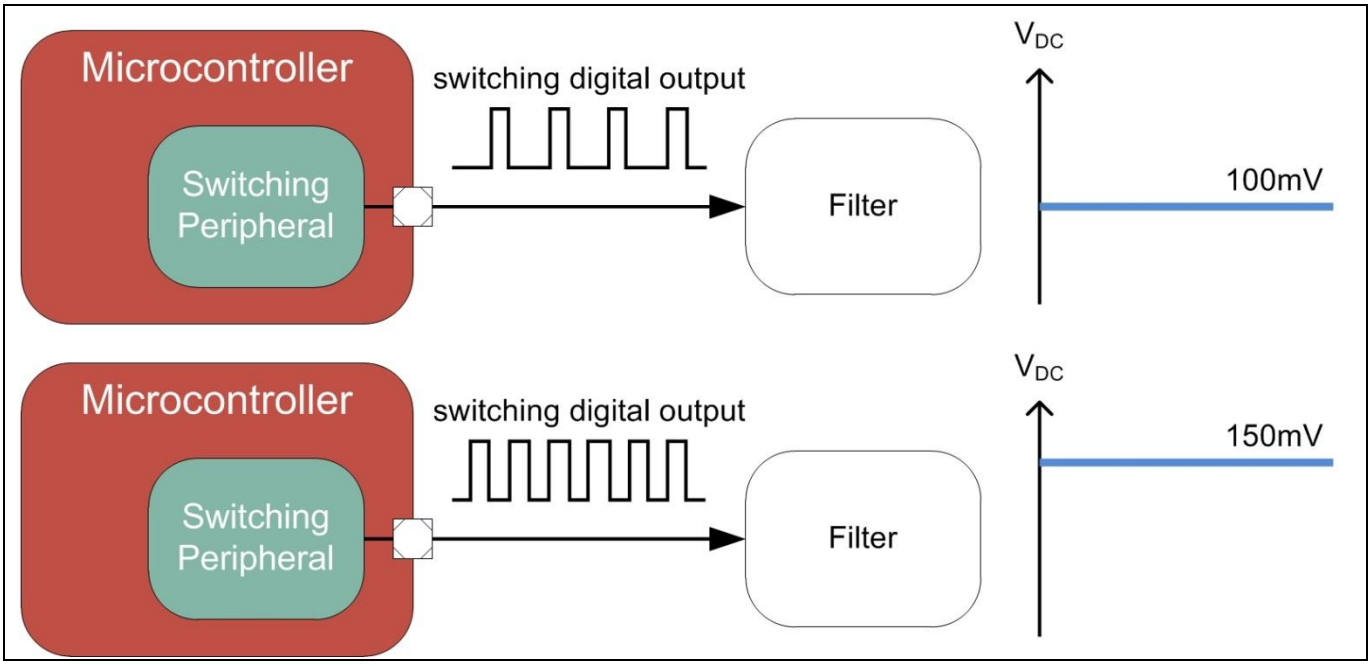

Figure 2 Pseudo DAC theory of application with PDM signal

In the XMC1000 Microcontroller family, the PWM pseudo DAC can be implemented using either the CCU4 or CCU8 and the PDM pseudo DAC can be implemented using the BCCU. The compare register value of the CCU4 or CCU8 determines the duty cycle of the PWM signal. In the case of the BCCU, the channel intensity register value determines the pulse density of the PDM signal.

1.2 Circuit diagram and signals

To achieve DAC conversion, the output of the BCCU or CCU4 or CCU8 must be configured in push-pull mode so that it is connected to an internal pull-up resistor. The RC low pass filter can be added externally (Figure 3).

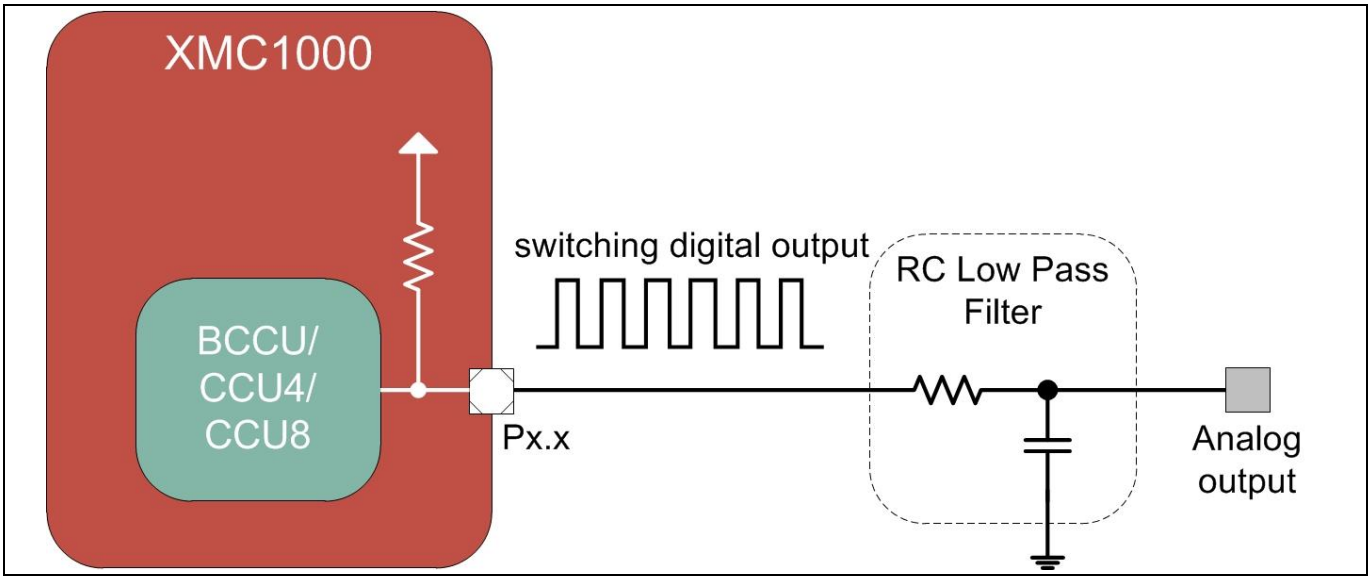

Figure 3 Low pass RC circuit to attenuate high frequency

Cut-off frequency (f_{cutoff}) is the key to determining the resistor and capacitor value. As a rule of thumb, f_{cutoff} must be as low as possible to eliminate most of the high frequency components and pass through only the DC component. Therefore, f_{cutoff} can be calculated as following:

$$f_{cutoff} = \frac{1}{2\pi RC}$$

As an example, using a combination of R = 47 kΩ and C = 1 µF,

$$f_{cutoff} = \frac{1}{2\pi \times 47 \times 10^{3} \times 1 \times 10^{-6}} \approx 3.4\ \text{Hz}$$

1.3 Calculating register value for desired reference voltage

This section provides tips on calculating the value to be programmed into the register to get the desired reference voltage.

1.3.1 Using the BCCU channel

The channel intensity value can be calculated using the following formula:

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

As an example, to generate a reference voltage of 100 mV in an operating voltage of 5 V:

$$Intensity = \frac{0.1 \times 4095}{5} = 82$$

1.3.2 Using the CCU4 or CCU8

The CCU4 or CCU8 compare value can be calculated using the following formula:

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

Section 3.1 covers how the period value can be derived.

As an example, to generate a reference voltage of 200 mV in an operating voltage of 3.3 V using a CCU4 timer with period value 639 (or 27 F_H):

$$Compare = \left(1 - \left(\frac{200}{3300}\right)\right) \times (639 + 1) = 601$$

2 Example use case 1: generating a reference level for analog comparator (ACMP)

In this simple example, the BCCU channel is used to generate a reference level for the ACMP slice in a current sensing application (Figure 4). The output pin of the BCCU channel is physically connected to the input pin of the ACMP. It is recommended to choose the BCCU channel output pin closest to the comparator input pin to keep the noise from the modulation signal as low as possible.

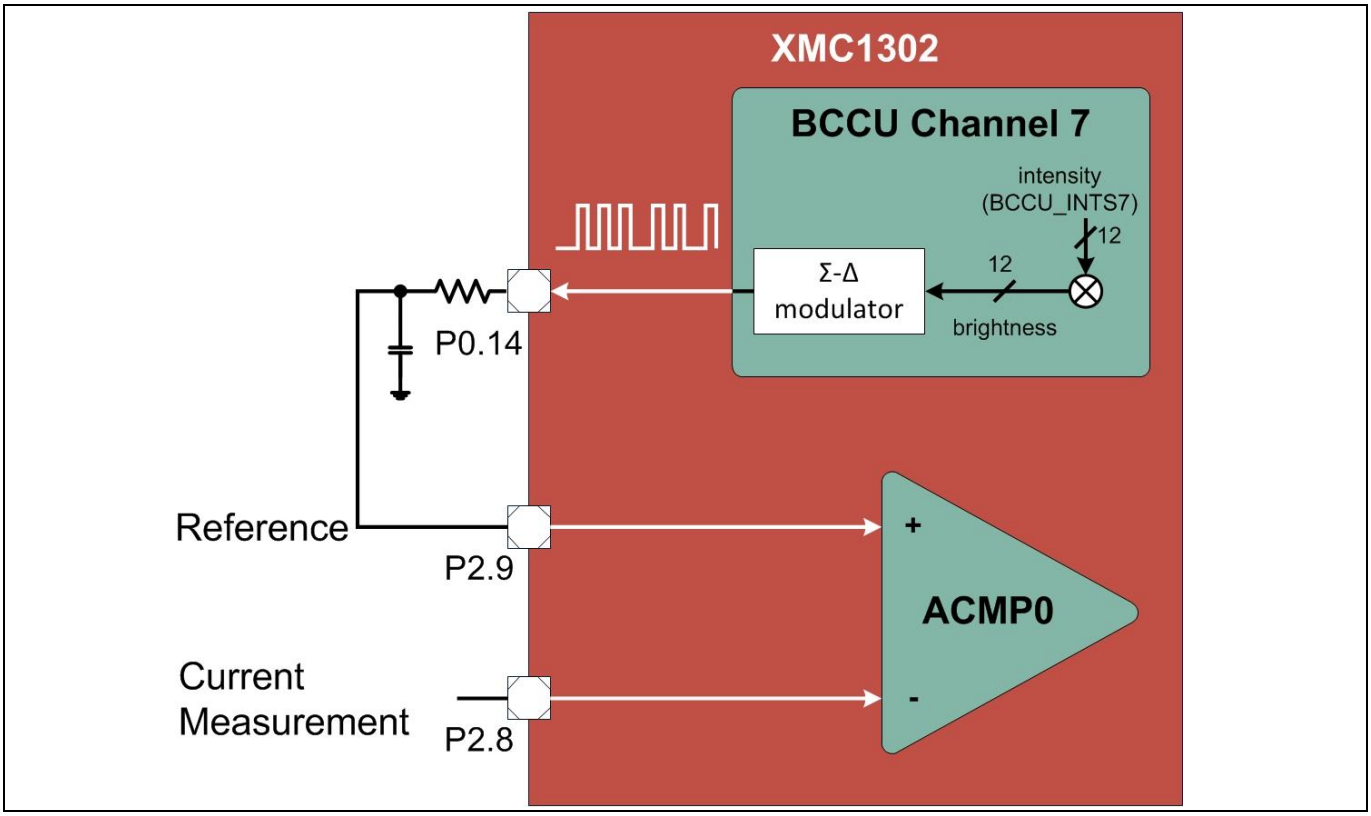

Figure 4 ACMP reference generation using BCCU channel

2.1 BCCU channel configuration

The dimming input to the BCCU channel is bypassed (Figure 4). This means that the brightness value is determined solely by the channel intensity value. The channel intensity can be adjusted via the register *INTSy*, where *y* is the channel value. The packer is disabled so that changes to the channel intensity can have an immediate effect on the PDM output.

2.2 Implementation using XMC™ lib

This section describes how the Infineon XMC™ lib can be used to configure a BCCU channel as a pseudo DAC for generating a reference value for the ACMP.

Configuration

The configuration section can be broken down to four parts:

1. System clock configuration:
   The peripheral clock (PCLK) frequency is configured to 64 MHz.

```
XMC_SCU_CLOCK_CONFIG_t clock_config =
{
  .pclk_src = XMC_SCU_CLOCK_PCLKSRC_DOUBLE_MCLK,
  .fdiv = 0,
  .idiv = 1
};
```

2. BCCU global configuration:
   This constitutes the configuration of BCCU clocks. 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 get a frequency value of 292.237 kHz.

```
XMC_BCCU_GLOBAL_CONFIG_t bccu_global_config =
{
  .fclk_ps = 0x50, //800 kHz @PCLK = 64 MHz
  .dclk_ps = 0xdb, //292.237 kHz @PCLK = 64 MHz
  .bclk_sel = XMC_BCCU_BCLK_MODE_NORMAL //200 kHz @PCLK = 64 MHz
};
```

3. BCCU channel configuration:
   This constitutes the configuration of the channel as described in section 2.1.

```
XMC_BCCU_CH_CONFIG_t dac_bccu_config =
{
    .pack_en = 0, // disable packer
    .dim_bypass = 1, // bypass dimming input
    .gate_en = 0, // disable gating
    .flick_wd_en = 0 // disable flicker watchdog
};
```

4. GPIO configuration:
   This constitutes the configuration of the port pin for the BCCU channel output.

```
XMC_GPIO_CONFIG_t dac_output_config =
{
    .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT4,
    .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW
};
```

Initialization

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

1. System clock initialization:

```
/* Ensure clock frequency is set at 64 MHz (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. BCCU channel initialization:
   The BCCU channel that is used is BCCU0_CH7.

```
/* init channel */
XMC_BCCU_CH_Init(BCCU0_CH7, &dac_bccu_config);
```

   The linear walk time is set to zero, for immediate effect.

```
/* Set linear walk time for PDMs*/
XMC_BCCU_CH_SetLinearWalkPrescaler(BCCU0_CH7, 0);
```

   The channel is then enabled. The mask required to enable channel 7 is 0x080.

```
/* Enable BCCU channel*/
XMC_BCCU_ConcurrentEnableChannels(BCCU0, 0x080);
```

   The reference voltage of 100 mV can then be initialized by setting the target channel intensity with the calculated value as in section 1.3.1.

```
XMC_BCCU_CH_SetTargetIntensity(BCCU0_CH0, 82);
```

   To effect the change on the BCCU channel output, start the linear walk. The same mask can be used.

```
XMC_BCCU_ConcurrentStartLinearWalk(BCCU0, 0x080);
```

4. GPIO initialization:

```
/* Enable BCCU PDM output */
XMC_GPIO_Init(P0_14, &dac_output_config);
```

2.3 Implementation using DAVE™ APP

This section describes how the COMP_REF APP (Figure 5) in DAVE™ can be used to implement this example.

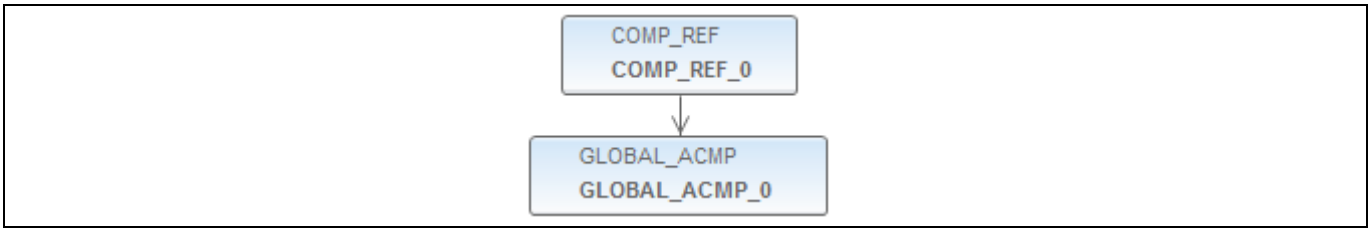

Figure 5 COMP_REF APP in DAVE™

The CMP_REF APP UI provides configurations for the ACMP slice such as reference source, hysteresis, glitch-filter, output inversion and option to connect the ACMP output to another peripheral via the Event Request Unit (ERU). The user must select the BCCU as the reference source (Figure 6) and the user can then configure the desired reference voltage and the operating voltage in the UI (Figure 7).

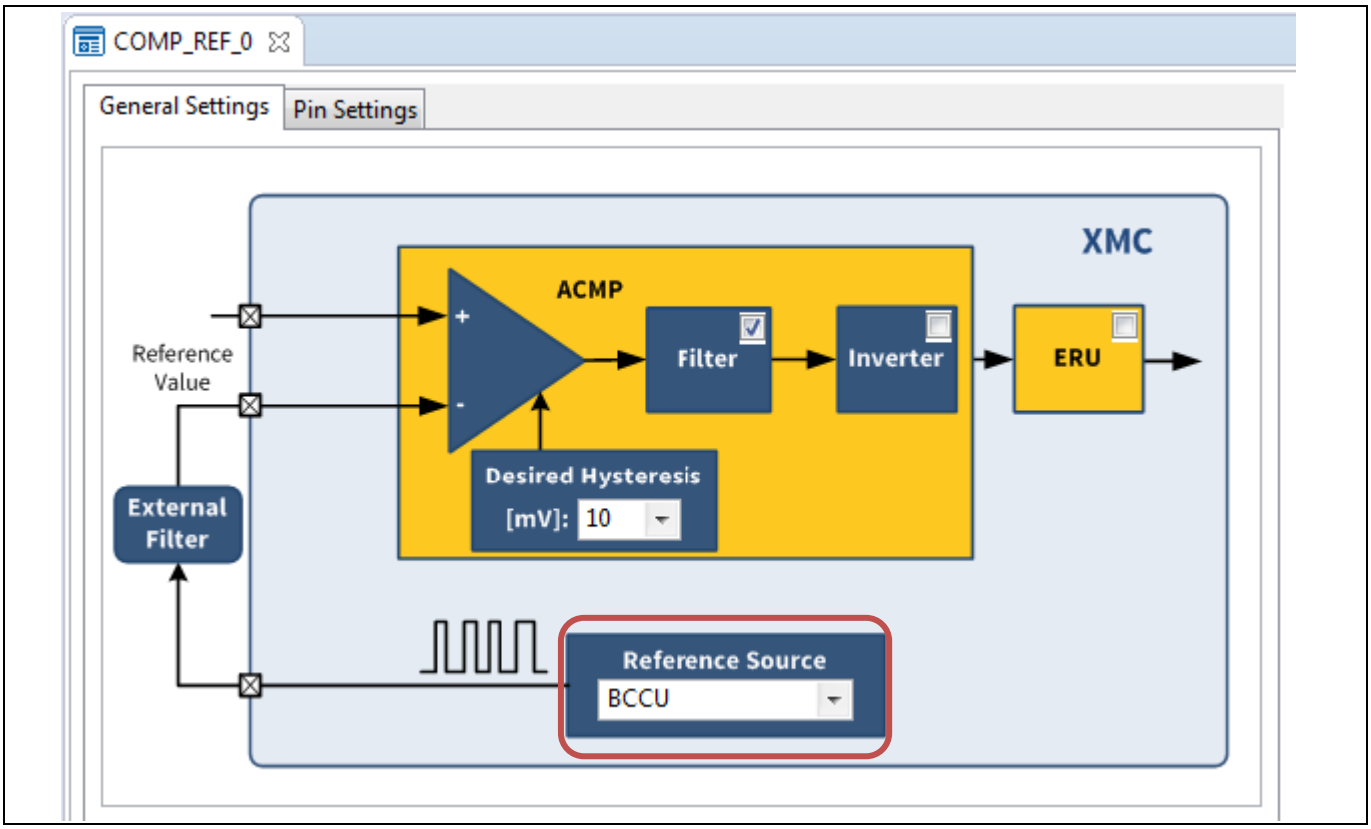

Figure 6 Select reference source in COMP_REF APP UI

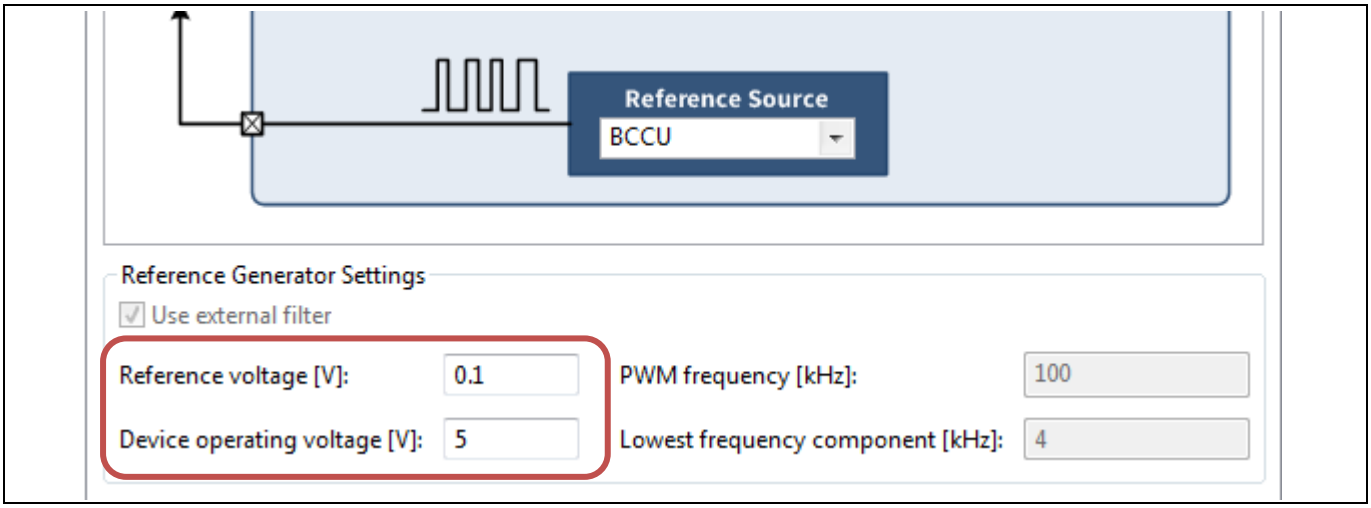

Figure 7 Configure reference and operating voltages

Pin assignment

The targeted ACMP slice and BCCU channel can be selected by assigning the respective pins (Figure 8).

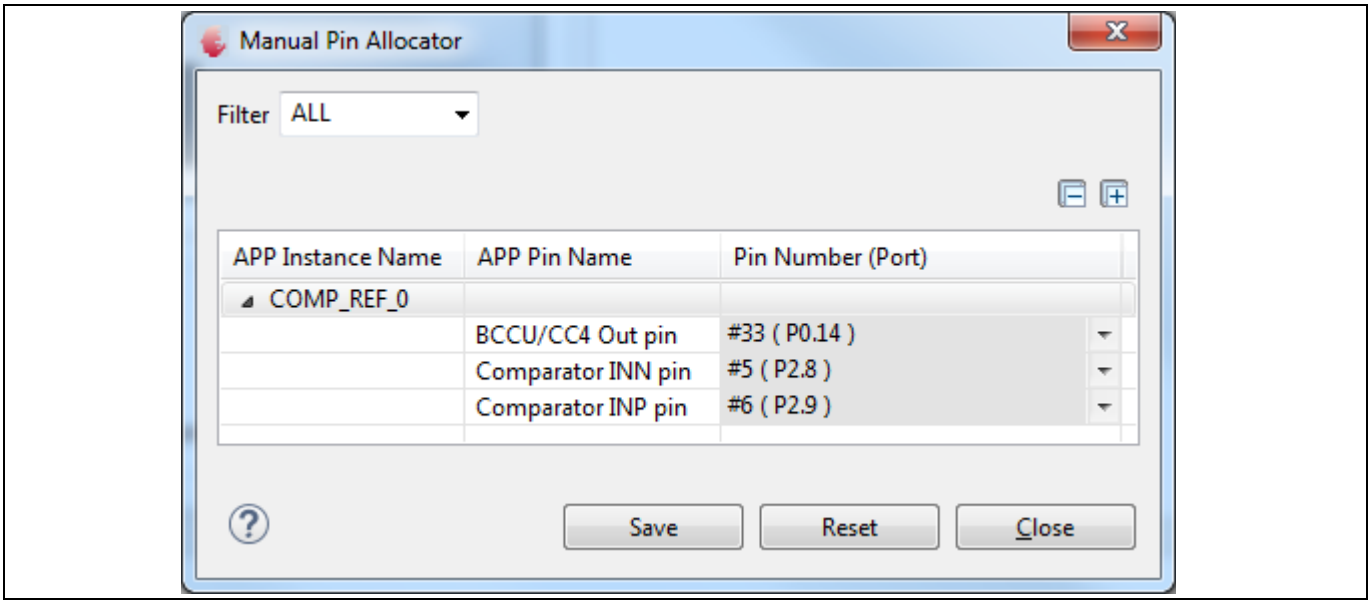

Figure 8 Manual assignment of pins

Updating reference value in-application

The reference value can be updated in-application using the Application Program Interface (API) provided by the COMP_REF APP, `COMP_REF_UpdateReferenceValue()`.

Note: This API accepts the reference value in mV.

The following is an example of updating the reference with a value of 2314 mV:

```
status = COMP_REF_UpdateReferenceValue(&COMP_REF_0, 2314);
```

3 Example use case 2: generating a sinusoidal waveform

In this example, the CCU4 timer is used to generate a sinusoidal waveform of 1 kHz (Figure 9).

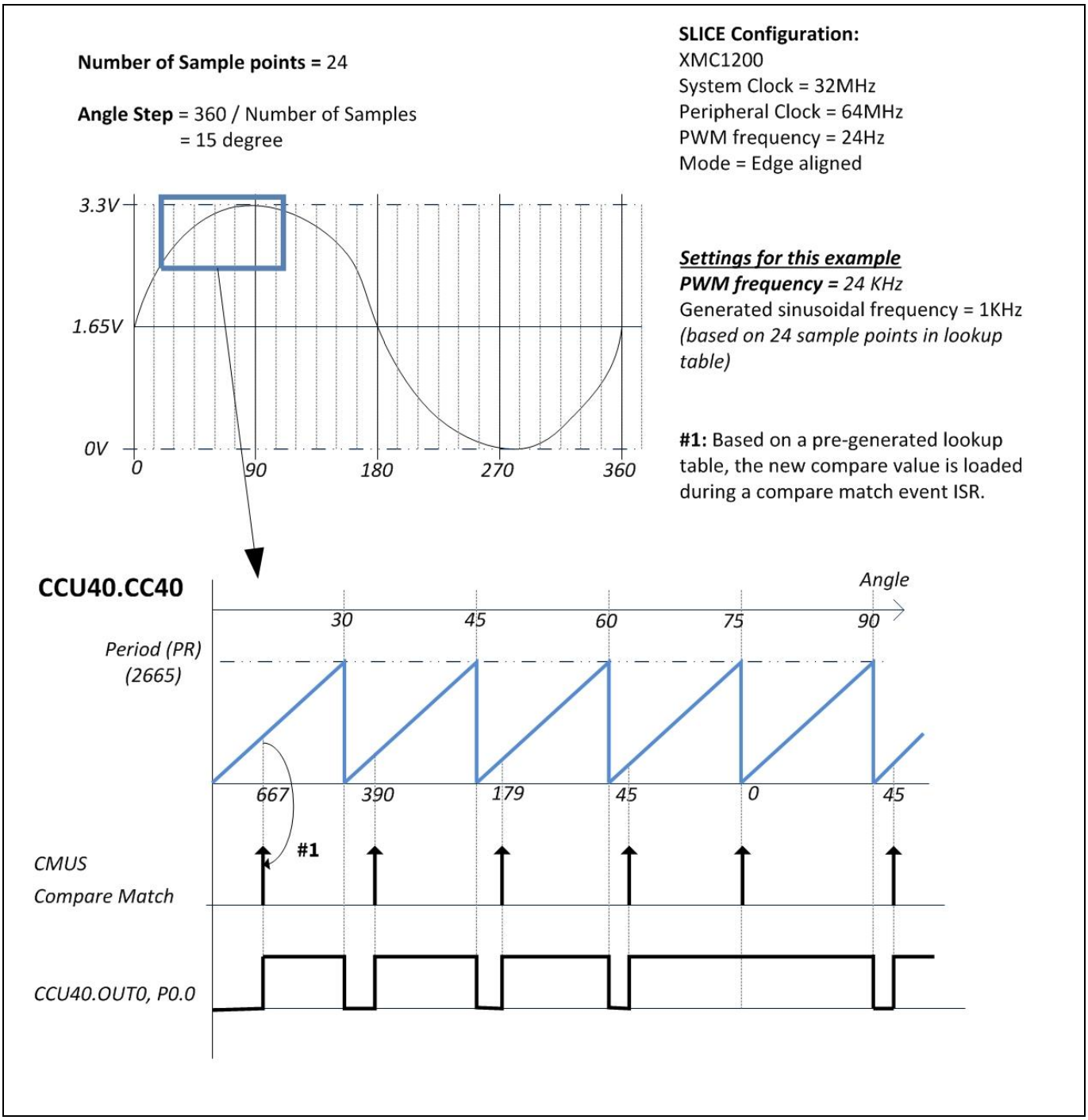

Figure 9 Example: generating a sinusoidal waveform

3.1 Deriving the period value

In this example, the frequency of the sinusoidal waveform generated is fixed at 1 kHz with 24 sample points. Therefore, the PWM frequency is fixed at 24 kHz. The clock relationship between f_{PWM}, f_{tclk} and f_{ccu4} is calculated as shown below:

- f_{ccu4} is the frequency of the CCU4 peripheral clock. It is the input to the PWM module.
- f_{tclk} is the timer resolution used to increment a timer counter. Each timer slice supports a dedicated prescaler value selector.
- In order for, f_{PWM}, frequency of the PWM signal, to be 24 kHz, the CCU4_CC40.PRS register is loaded with value 2667.

Timer frequency: $f_{tclk} = \frac{f_{ccu4}}{Prescaler}$

Period value: $CCU4_{CC40}.PRS = \frac{f_{tclk}}{f_{PWM}} - 1$

Table 1 Calculated prescaler factor and period values

Type	Calculated value
Prescaler value	$2^0 = 1$
Period @ 24 kHz frequency	2665

3.2 Generating a look-up table

DAC resolution is the smallest increment in the analog output voltage that corresponds to a single increment in the DAC digital count. In other words, the finest increment of output voltage level is directly proportional to incrementing the CCU4 PWM duty cycle value.

In general, the resolution increases with the increase of sample points in the PWM signal. In this example, the sinuoisodal waveform is divided into 24 sample points (Figure 10).

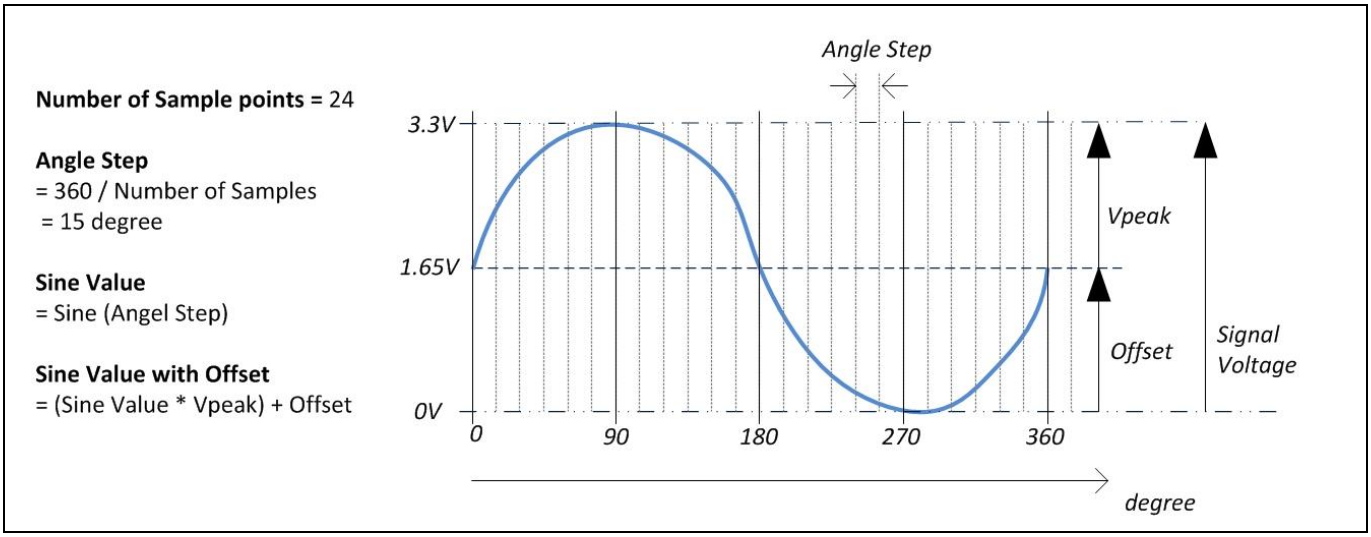

Figure 10 Deriving the sine value with reference to signal voltage

Each change in PWM duty cycle is the equivalent of one DAC sample.The CCU4 period is fixed at 24 kHz. The calculation for the look-up table is as shown below:

Signal frequency: Signal frequency = $f_{PWM} * number\ of\ samples$

Duty cycle: $DC = \frac{sine\ value\ with\ offset}{signal\ voltage}$

Compare value: $CCU4_{CC40}.CRS = (1 - DC) * (PRS + 1)$

Table 2 Calculated look-up table for compare values

Angle step (degree)	Sine value	Sine value with offset	Duty cycle (%)	Compare value (CRS)
0 or 360	0.000	1.650	50.00	1333
15	0.259	2.077	62.94	988
30	0.500	2.475	75.00	667
45	0.707	2.817	85.36	390
60	0.866	3.079	93.30	179
75	0.966	3.244	98.30	45
90	1.000	3.300	100.00	0
105	0.966	3.244	98.30	45
120	0.866	3.079	93.30	179
135	0.707	2.817	85.36	390
150	0.500	2.475	75.00	667
165	0.259	2.077	62.94	988
180	0.000	1.650	50.00	1333
195	-0.259	1.223	37.06	1678
210	-0.500	0.825	25.00	2000
225	-0.707	0.483	14.64	2276
240	-0.866	0.221	6.70	2487
255	-0.966	0.056	1.70	2621
270	-1.000	0.000	0.00	2666
285	-0.966	0.056	1.70	2621
300	-0.866	0.221	6.70	2487
315	-0.707	0.483	14.64	2276
330	-0.500	0.825	25.00	2000
345	-0.259	1.223	37.06	1678

3.3 Implementation with XMC™ lib

3.3.1 Macro and variable settings

XMC™ lib project includes:

```
#include <xmc_ccu4.h>
#include <xmc_gpio.h>
#include <xmc_scu.h>
```

Project macro definitions:

```
#define MODULE_PTR        CCU40
#define MODULE_NUMBER     (0U)

#define SLICE0_PTR        CCU40_CC40
```

```
#define SLICE0_NUMBER     (0U)
#define SLICE0_OUTPUT     P0_0
```

Project variables definition:

```
volatile uint8_t count=0;
uint16_t comparevalue[24]=      /* sine table for duty cycle*/
{
             1333U,
             988U,
             667U,
             391U,
             179U,
             45U,
             0U,
             45U,
             179U,
             391U,
             667U,
             988U,
             1333U,
             1678U,
             2000U,
             2276U,
             2488U,
             2621U,
             2667U,
             2621U,
             2488U,
             2276U,
             2000U,
             1678U
};
```

3.3.2 XMC™ lib peripheral configuration structure

XMC System Clock Unit (SCU) configuration:

PWM period is calculated based on PCLK which is equivalent to 64 MHz.

```
XMC_SCU_CLOCK_CONFIG_t clock_config =
{
  .pclk_src = XMC_SCU_CLOCK_PCLKSRC_DOUBLE_MCLK,
  .rtc_src = XMC_SCU_CLOCK_RTCCLKSRC_DCO2,
  .fdiv = 0,
  .idiv = 1
};
```

XMC™ Capture/Compare Unit 4 (CCU4) configuration:

```
XMC_CCU4_SLICE_COMPARE_CONFIG_t SLICE0_config =
{
  .timer_mode = (uint32_t) XMC_CCU4_SLICE_TIMER_COUNT_MODE_EA,
  .monoshot = (uint32_t) false,
  .shadow_xfer_clear = (uint32_t) 0,
```

```
  .dither_timer_period = (uint32_t) 0,
  .dither_duty_cycle = (uint32_t) 0,
  .prescaler_mode = (uint32_t) XMC_CCU4_SLICE_PRESCALER_MODE_NORMAL,
  .mcm_enable = (uint32_t) 0,
  .prescaler_initval = (uint32_t) 0,   /* in this case, prescaler = 2^10 */
  .float_limit = (uint32_t) 0,
  .dither_limit = (uint32_t) 0,
  .passive_level = (uint32_t) XMC_CCU4_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .timer_concatenation = (uint32_t) 0
};
```

XMC™ GPIO configuration:

```
XMC_GPIO_CONFIG_t SLICE0_OUTPUT_config =
{
   .mode = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT4,
   .input_hysteresis = XMC_GPIO_INPUT_HYSTERESIS_STANDARD,
   .output_level = XMC_GPIO_OUTPUT_LEVEL_LOW,
};
```

3.3.3 Interrupt service routine function implementation

The CCU40 interrupt handler function is created to update the timer compare match values to achieve a sine signal.

```
void CCU40_0_IRQHandler(void)
{
      /* Clear pending interrupt */
      XMC_CCU4_SLICE_ClearEvent(SLICE0_PTR,XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP);

      /* Set new duty cycle value */
      XMC_CCU4_SLICE_SetTimerCompareMatch(SLICE0_PTR, comparevalue[count]);

      count++;
      if(count==24)
      {
        count=0;
      }
      XMC_CCU4_EnableShadowTransfer(MODULE_PTR, XMC_CCU4_SHADOW_TRANSFER_SLICE_0);
}
```

3.3.4 Main function implementation

Before the start and execution of timer slice software for the first time, the CCU4 must be initialized appropriately using the following sequence:

- Set up the system clock

```
  /* Initializes the clock settings */
  XMC_SCU_CLOCK_Init(&clock_config);
```

- Enable clock, enable prescaler block and configure global control

```
/* Enable clock, enable prescaler block and configure global control */
XMC_CCU4_Init(MODULE_PTR, XMC_CCU4_SLICE_MCMS_ACTION_TRANSFER_PR_CR);
/* Start the prescaler and restore clocks to slices */
XMC_CCU4_StartPrescaler(MODULE_PTR);

/* Start of CCU4 configurations */
/* Ensure fCCU reaches CCU40 */
XMC_CCU4_SetModuleClock(MODULE_PTR, XMC_CCU4_CLOCK_SCU);
```

- Configure slice(s) functions, interrupts and start-up

```
/* Initialize the slice */
XMC_CCU4_SLICE_CompareInit(SLICE0_PTR, &SLICE0_config);

/* Program 24 kHz frequency */
XMC_CCU4_SLICE_SetTimerCompareMatch(SLICE0_PTR, comparevalue[count]);
XMC_CCU4_SLICE_SetTimerPeriodMatch(SLICE0_PTR, 2665U);

/* Enable shadow transfer */
XMC_CCU4_EnableShadowTransfer(MODULE_PTR, \
          (uint32_t)(XMC_CCU4_SHADOW_TRANSFER_SLICE_0| \
                     XMC_CCU4_SHADOW_TRANSFER_PRESCALER_SLICE_0));

/* Enable compare match events */
XMC_CCU4_SLICE_EnableEvent(SLICE0_PTR, XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP);

/* Connect compare match event to SR0 */
XMC_CCU4_SLICE_SetInterruptNode(SLICE0_PTR, \
   XMC_CCU4_SLICE_IRQ_ID_COMPARE_MATCH_UP, XMC_CCU4_SLICE_SR_ID_0);

/* Set NVIC priority */
NVIC_SetPriority(CCU40_0_IRQn, 3U);

/* Enable IRQ */
NVIC_EnableIRQ(CCU40_0_IRQn);

/* Enable CCU4 PWM output */
XMC_GPIO_Init(SLICE0_OUTPUT, &SLICE0_OUTPUT_config);

/* Get the slice out of idle mode */
XMC_CCU4_EnableClock(MODULE_PTR, SLICE0_NUMBER);
```

- Start timer running

```
/* Start the timer */
XMC_CCU4_SLICE_StartTimer(SLICE0_PTR);
```

Revision history

Major changes since the last revision

Page or reference	Description of change

Edition <2015-12-01>

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference
AN_201511_PL30_0011

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.