XMC1000, XMC4000

32-bit microcontroller series for industrial applications

Capture Compare Unit 8 (CCU8)

AP32288

Application Note

About this document

Scope and purpose

This application note provides a brief introduction to the key features of the Capture Compare Unit (CCU8) module and some typical application examples. It also provides hints for users who wish to use the CCU8 to develop motor control applications with the XMC™ microcontroller family.

Intended audience

This document is intended for engineers who are familiar with the XMC™ microcontroller family.

Applicable products

- XMC1000
- XMC4000
- DAVE ™

References

The user's manual can be downloaded from http://www.infineon.com/XMC.

DAVE™ and its resources can be downloaded from http://www.infineon.com/DAVE

Table of contents

Table of contents

1 Introduction to the CCU8 basic features

1.1 CCU8 basics

The CAPCOM8 is a multi-purpose timer unit for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. It is designed with repetitive structures with multiple timer slices that have the same base functionality. The internal modularity of the CCU8 translates into a software friendly system for fast code development and portability between applications.

The following image shows the main functional blocks of one of the four CC8y slices on a CCU8x.

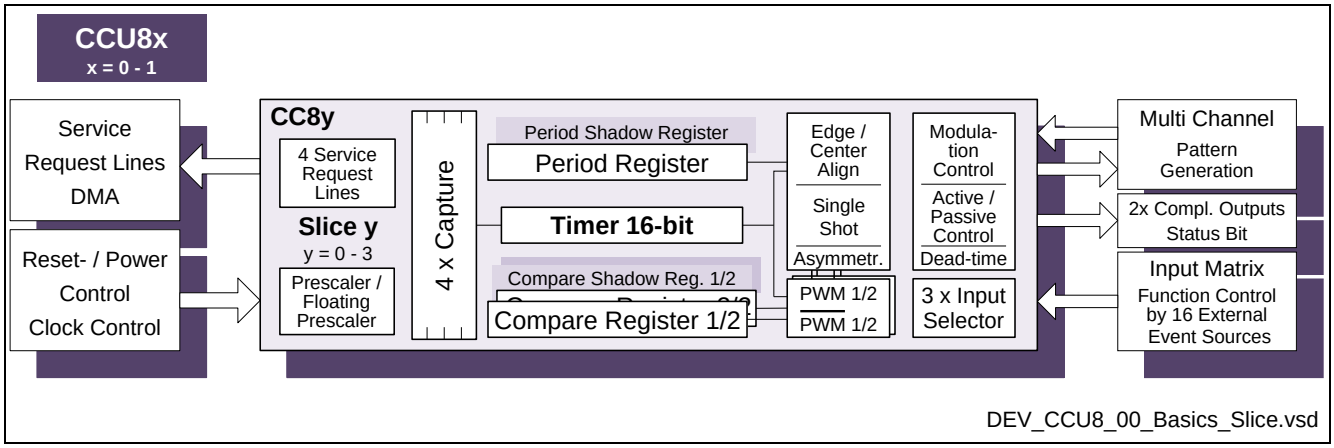


Figure 1 The timer slice block diagram

1.2 Basic timer functions

Each timer slice can handle all the basic modes and the typical options illustrated in Figure 2 below.

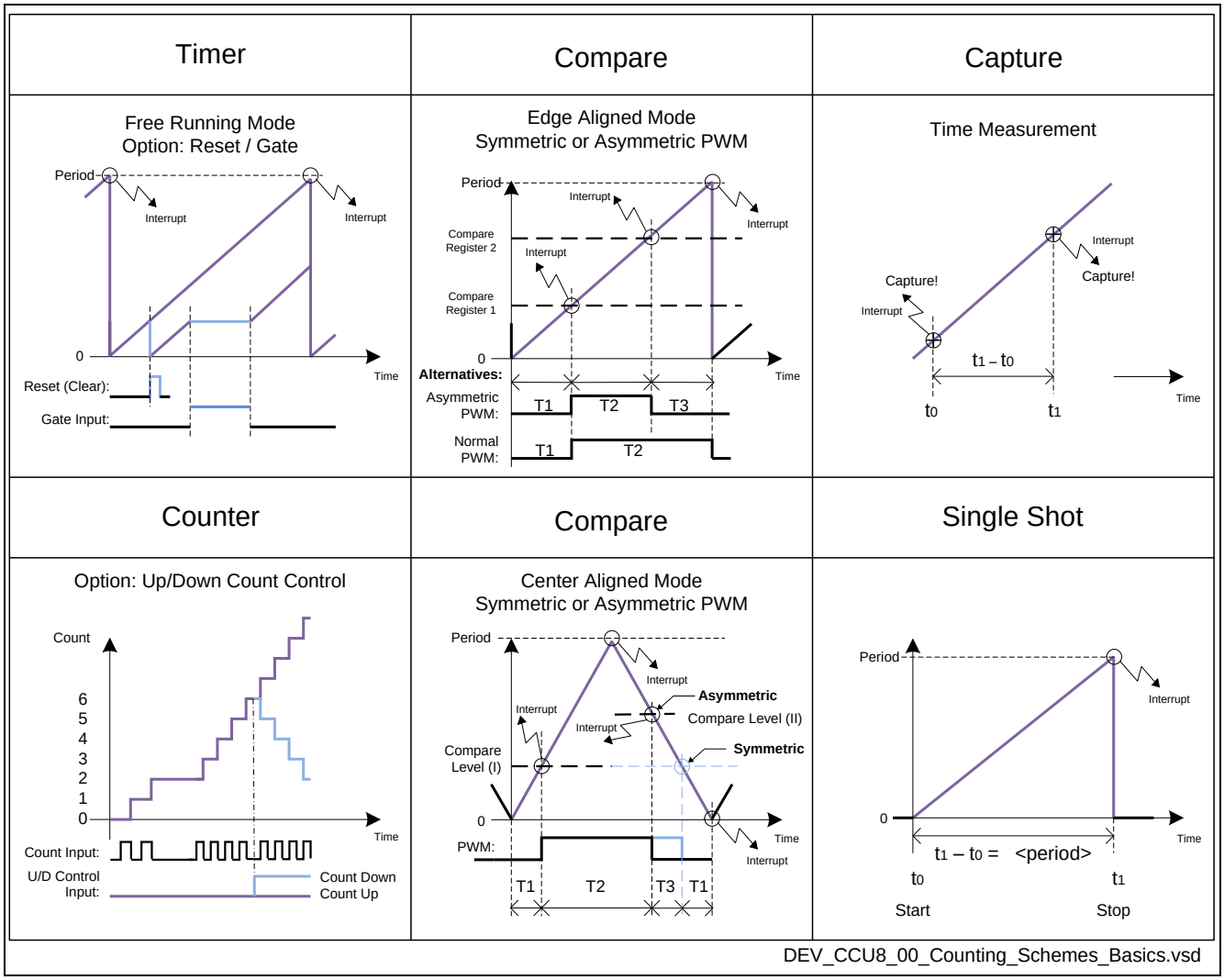


Figure 2 Basic functions of each timer slice

1.3 The compound CAPCOM8 system

Each CCU8x has four 16-bit timer slices CC8y (y=3-0), which can be concatenated up to 64-bit.

A slice has:

- 1 timer
- 4 capture registers
- 1 period register
- 2 compare registers

Both the period and compare registers have shadow registers. Each slice can work independently in different modes, but they can also be synchronized, even to other CCU8 slices. They perform multi-channel/multi-phase pattern generation with parallel updates.

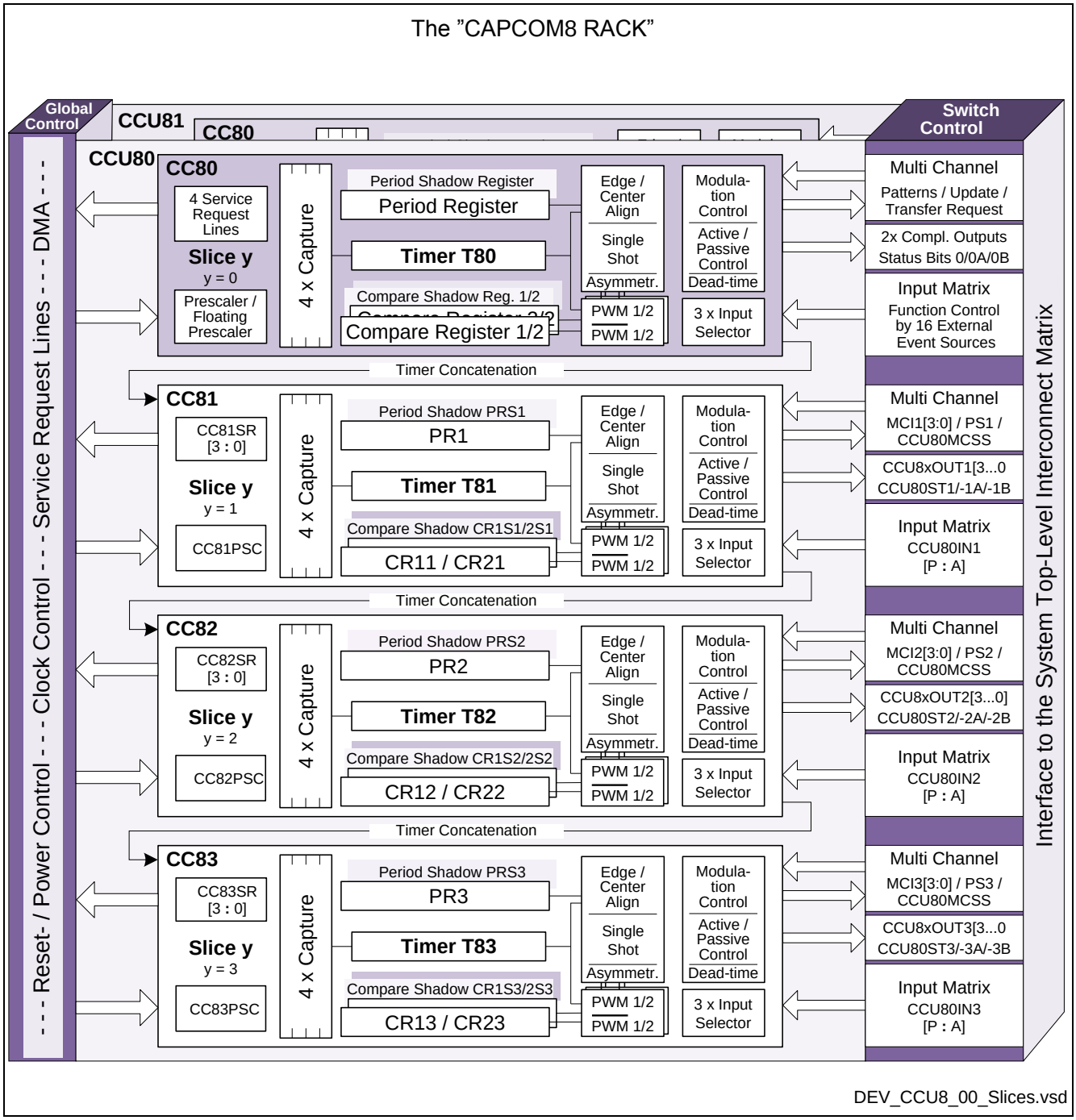


Figure 3 The basic capture/compare unit system of CAPCOM8

1.4 CCU8 applications

Here are some typical example applications that demonstrate the various capabilities of the CAPCOM timer slices of the CCU8:

1. Simple time base with synchronization option by external events control
2. Power conversion system (PFC, SMPS) using single shot mode

3. Feedback sensor event monitoring and revolution by capture, count and position interface facilities (POSIF)
4. Multi-signal pattern on output pins, created by parallel multi-channel control
5. Drive & motor control with multi-phase system, phase adjustment and trap handling
6. 3-Level PWM for inverters and Direct Torque Control (DTC) of AC motors and high precision synchronous motors
7. External events control of timer input functions by requests from external system units
8. Dithering PWM or period for DC-level precision, reduced EMI, fractional split of Periods into micro step
9. Auto adjusting time base by floating prescaler for adaption of time measurement to a wide range of dynamics

The same applications are illustrated in the following figure:

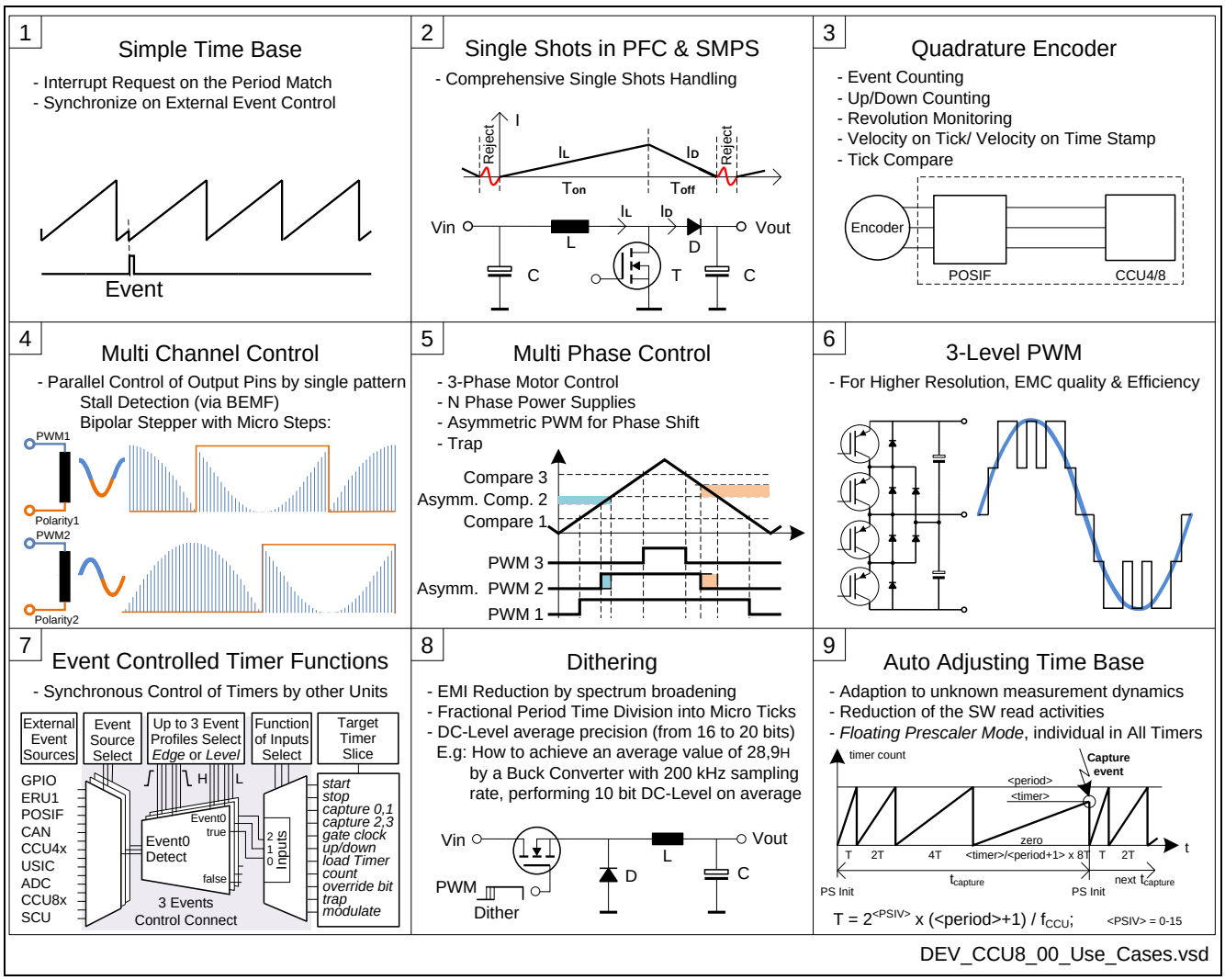


Figure 4 Some features and applications illustrating an CAPCOM unit (CCU) features

1.5 Additional CCU8 features

Features	Operation
Single shot	If a slice is set in Timer Single Shot Mode (CC8yTC.TSSM), the timer and its run bit (TRB) are cleared by the period/one match that occurs next to when the TSSM bit was set. As a result, the timer stops running.
Timer concatenation	Any timer slice can be concatenated with an adjacent timer slice by setting CC8yTC.TCE = 1.
Dithering PWM	It can be used with very slow control loops that cannot update the period/compare values in a fast manner. The precision can be maintained on long runs.
Dithering period time	Micro ticks can be used in the Interpolation between sensor pulses to achieve higher precision position monitoring.
Floating prescaler	By changing of the timer clock frequency periodically (no compare/capture event), the dynamic range is autonomously adapted to any time length.
External modulation	The output pin signal of a slice is modulated by external events.
Output state override	An external input signal source can override a slice's status bit (CC8yST) on an edge event by other external input signal source.
Multi-channel cControl	The output state of timer slices PWM signal(s) can be controlled in parallel by a single pattern.
External load	Each slice of CCU8 allows the user to select an external signal as the trigger for reloading the timer value with current compare/period register value.
Trap function	This function forces the PWM output into a predefined state, preset in the active/passive PSL bit. This allows the power device to be safely switched off.

1.6 CCU8 input control

1.6.1 Synchronized control of CAPCOM units on external events

External events control distribution to CCUs (including CCU8) allows for synchronized timer control in advanced applications. For example, in motor drive and power control, where 3-level inverters might require 12 synchronized PWMs. The limits are the realizable topography or timing pattern complexity range.

1.6.2 External control basics

The input functions of a slice are controlled by external sources. The external source(s), active mode(s) and input function(s) should be mapped to the 3 inputs of the slice in the CC8yINS and CC8yCMC registers. Function mode extension alternatives can be added by selections in the CC8yTC timer slice control register.

1.6.3 External events control

An external event control request can be an edge or level event signal from a peripheral unit or a GPIO. It can be linked to the CCU8xCC8y slices input selection stages via a comprehensive matrix. A slice with any of its 3 events setups detects a considered source-event-input profile, can be function controlled "remotely" this way.

1.6.4 External event sources

CCU8xCC8y input functions can be linked to external trigger requests from sources such as: GPIO, ERU, POSIF, CAN, CCU4x, USIC, ADC, CCU8x or SCU. Pin connections are given by the top-level interconnect matrix and the CC8yINS[P:A] input select vector. The CC8yCMC register is used for the function selection.

1.6.5 External event input functions

There are 11 timer input functions (e.g. ‘Start the Timer’) eachcontrollable by external events via 3 selectable input lines with configurable source-event profile conditions to the timer slices CC8y (y=0-3) of a CCU8x unit for start, stop, capture0-3, gate, up/down, load, count, bit override, trap and modulate output control.

There are also some extended input functions in the register CC8yTC for extended start, stop with flush/start, flush/stop or flush or extended capture mode. Together, with a read access register (ECRD), these simplify administration of capture registers and full-flags when more than one slice is used in capture mode.

1.7 Capture basics

Each CAPCOM8 (CCU8x) has 4 timer-slices. Each slice has 4 capture value registers, split into 2 pairs that capture on the selected event control input: Capt0 or Capt1, according to 2 possible pair schemes: either as 2 pairs for different events respectively to Capt0 and Capt1, or cascaded for the same event via Capt1.

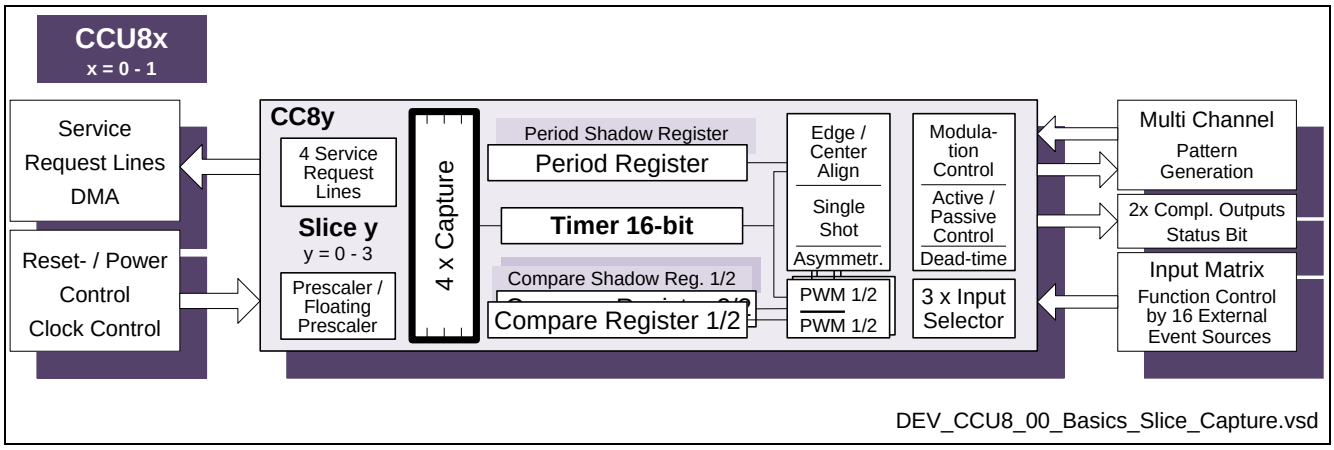


Figure 5 Timer slice with four capture registers

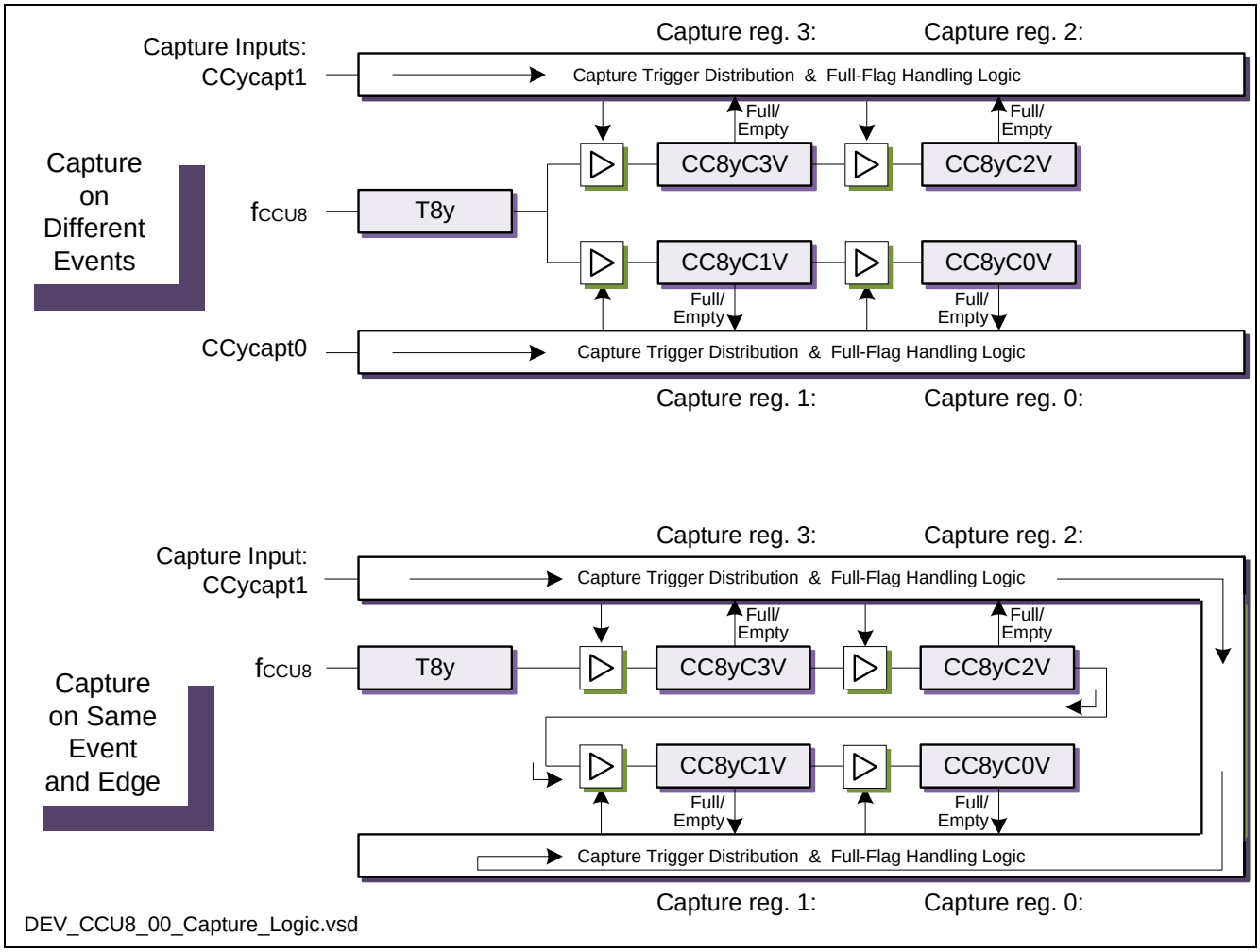


Figure 6 Basic capture mechanism – setup in two possible scheme alternatives

1.8 CCU8 output control

1.8.1 External control by timer events

A timer event can trigger external actions via the top-level interconnect matrix or on request for an interrupt. Each CAPCOM8 has four service request lines and each slice has a dedicated output signal CC8ySR[3...0], selectable to a line by CC8ySRS. This mean timer slice events can request direct peripheral actions or an interrupt.

1.8.2 Top-level control of event request to/from a timer slice

Top-level control also means conditional control of event requests between a slice and other action providers. The Event Request Unit (ERU1) and the top-level interconnect matrix can combine, control and link event signals according to user defined request-to-action event patterns. For example, invoke I/O states, time windowing etc.

1.9 Compare basics

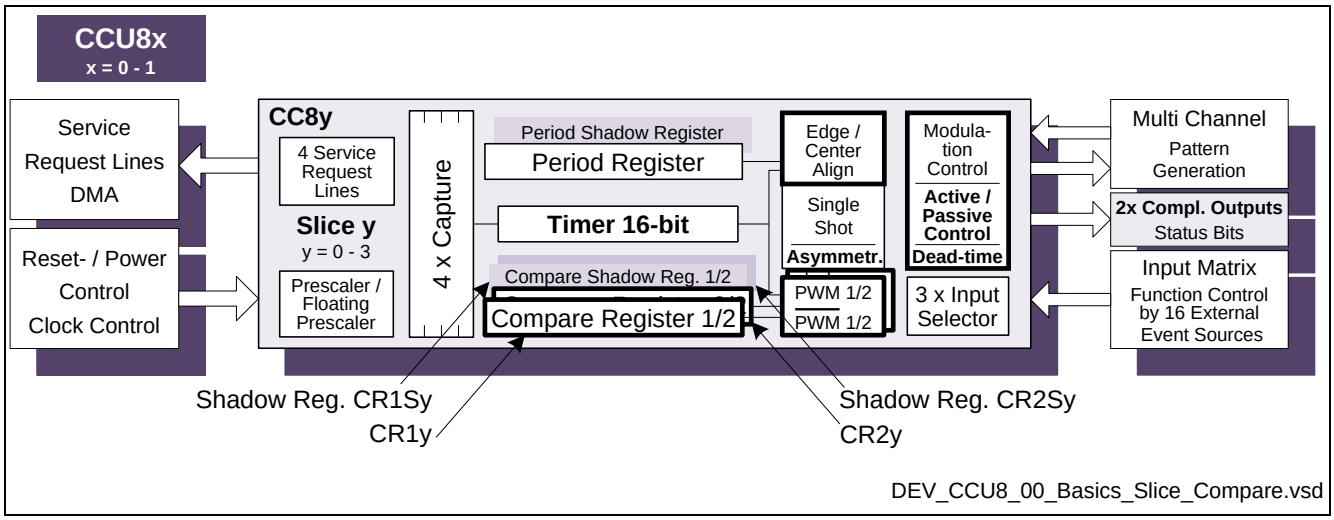


Figure 7 Timer slice compare registers and PWM related blocks

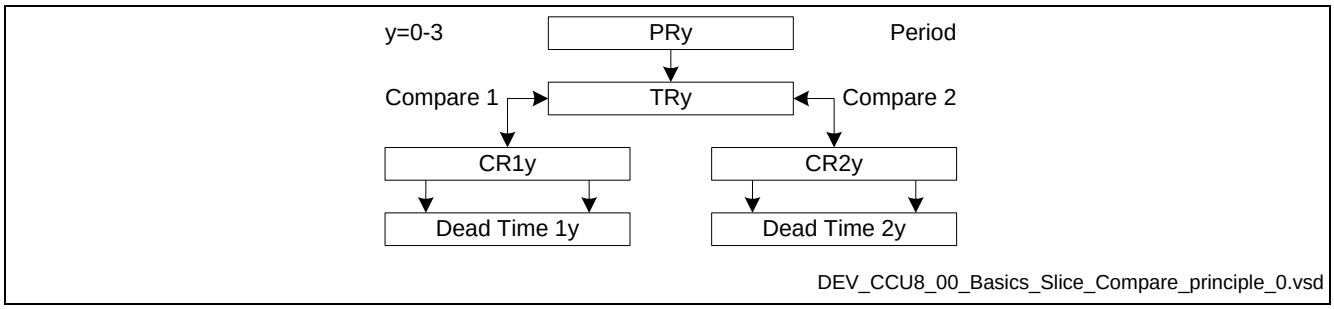


Figure 8 Basic blocks for Symmetric/Asymmetric PWM generation with dead-time

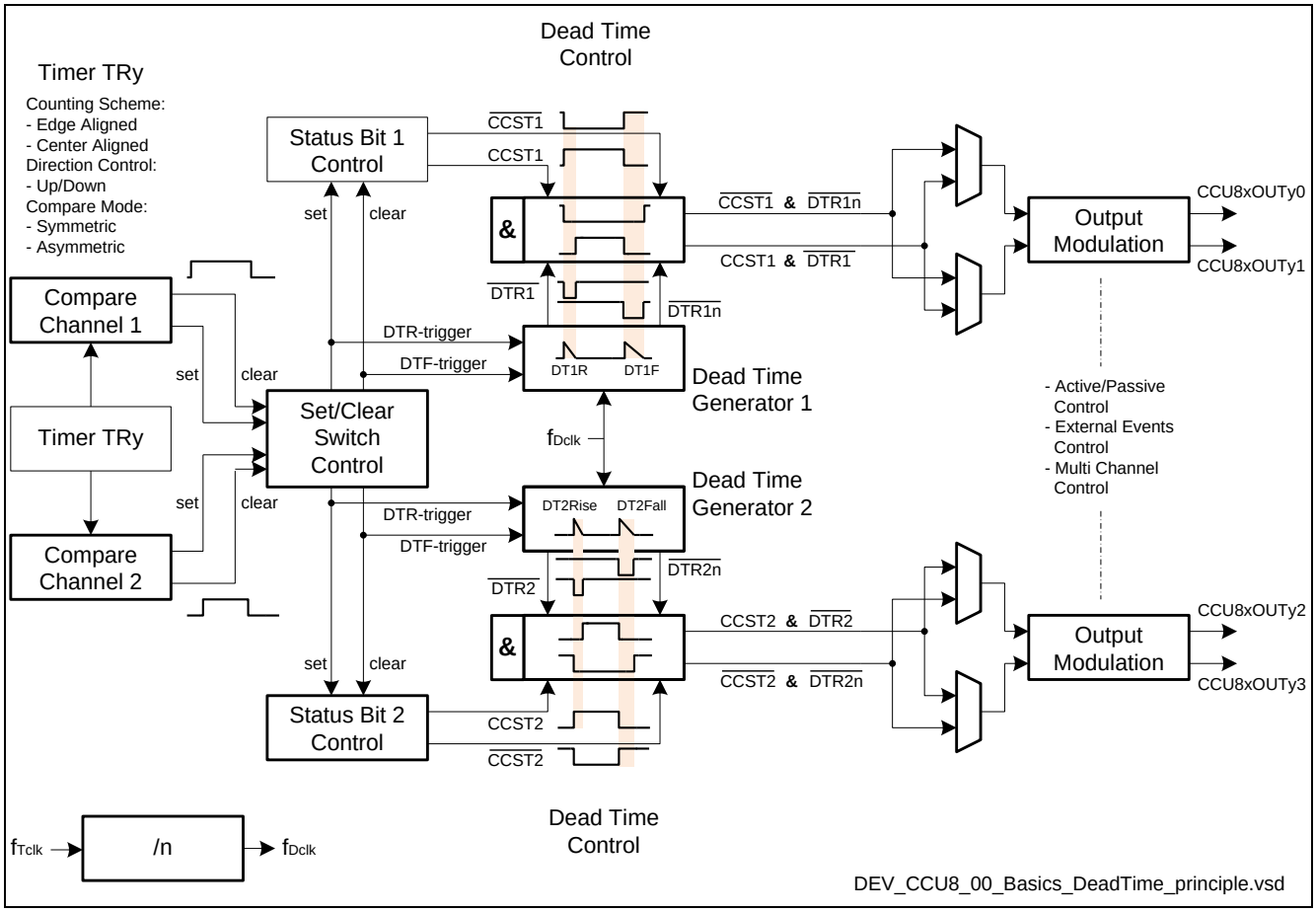


Figure 9 Dead-time generation principles

1.9.1 CCU8 shadow transfers

Whatever the slice configuration, whatever level of complexity, whatever the signal patterns, all the timer function parameters of the CAPCOM4 timers are assured coherent updates by hardware. They are updated from values in the shadow registers that, on a global preset request, are transferred simultaneously to all function registers at a period match or one match.

1.9.2 Shadow transfer of compare register values

There is one global register (GCSS) carrying all enable flags that have to be preset by software to selectively activate the targeted shadow transfer requests. It is also cleared by hardware after the transfer, to achieve total real time correctness.

The compare values that are targeted for an update operation have to be written into the CC8yCR1S/CR2S shadow registers AND the corresponding slice transfer set enable bits. For example SySE in GCSS, must be preset before period match (in edge aligned mode) or period/one match (in center aligned mode).

Beside the compare (CR1/CR2) values, there are also the timer Period Register (PR) and the PWM active/passive control bit (PSL) that can be updated simultaneously on the SySE flag. Dithering or floating prescaler values are able to get a simultaneous update via the SyDSE and SyPSE request flags.

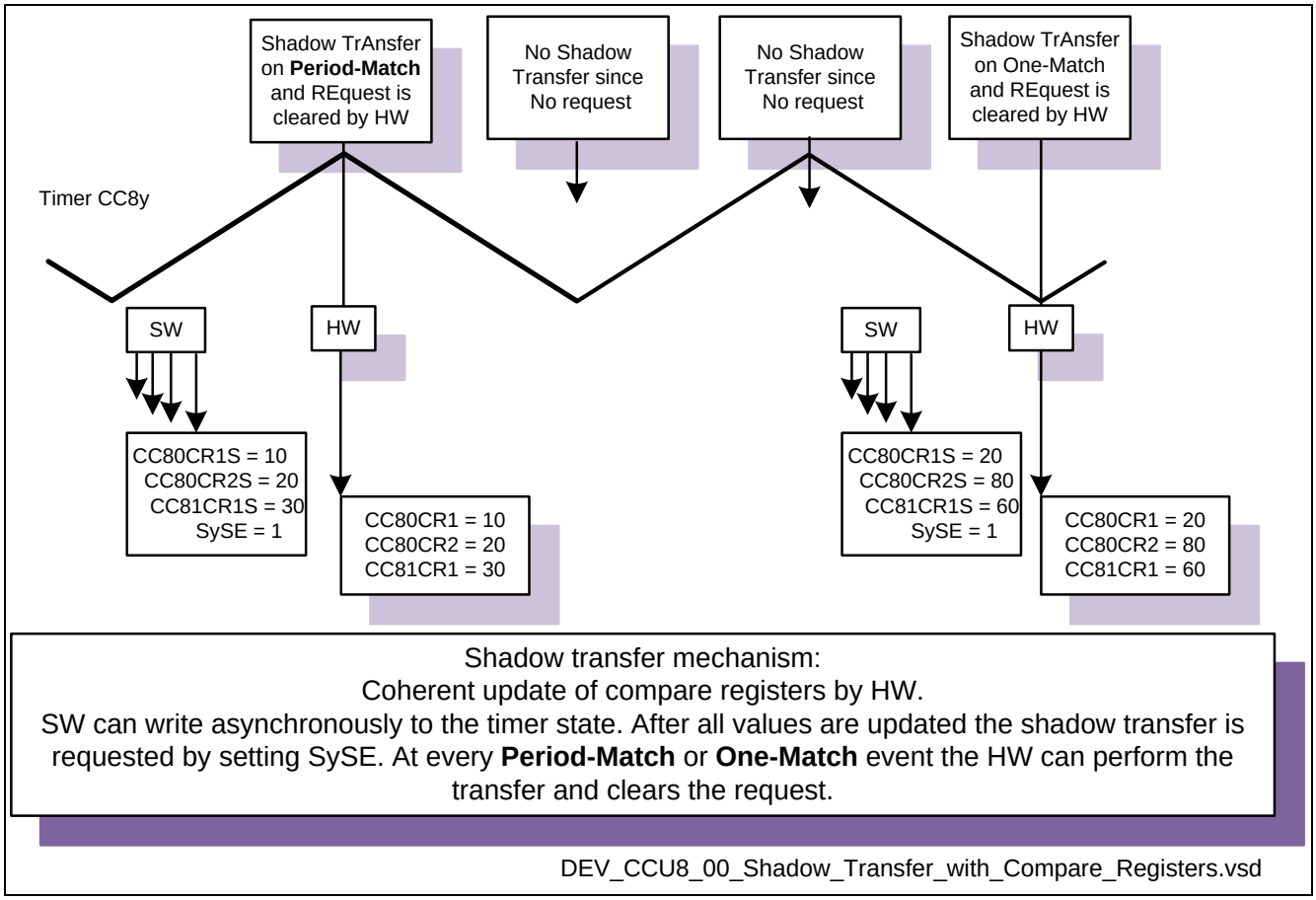


Figure 10 Basic shadow transfer mechanism for compare register values

1.9.3 CCU8 output state and output pin PASSIVE/ACTIVE level control

The PASSIVE/ACTIVE state of a slice's internal output CCUxSTy (status bit CC8yST) is controlled by the compare level and the external modulation mode. The CC8yPSL passive/active bit PSL controls whether the external output pin state CCU8xOUTy (for example, the PWM) should be passive low / active high or vice-versa.

1.10 How to start a timer

1.10.1 Initialization sequence

Before the start and execution of timer slice software for the first time, the CCU8 must be initialized appropriately using the following sequence:

- Apply reset
- Release reset
- Enable clock
- Enable prescaler block
- Configure global control
- Configure slice(s) functions, interrupts and start-up

1.10.2 Start-up enable

In the last part of the CCU8 initialization sequence the startup value(s) for a specific compare channel status of the timer slice(s) could be configured by the respective GCSS.SyTS bit. After that, the default IDLE mode has to be removed from the timer slice(s) in the GIDLC register and then start or global start can be initiated.

1.10.3 Start timer running

There are two ways to start a timer:

- Directly by software setting the Timer Run Bit Set (TRBS)
- Indirectly by hardware when a specific event occurs in an external unit as determined by the top-level connection matrix of external events control for CAN, ADC, USIC, IO, CCU4/8, ERU1, POSIF and so on.

1.10.4 Global start of CCU8

To achieve a synchronized start of both CAPCOM Units (CCU4x and CCU8x) use either

- A global start by software, with the CCUx global start control bits in the CCUCON global start control register
- A global start by hardware, indirectly with external events control using the CC8yINS and CC8yCMC registers.

1.10.5 Global start of the CCU4 and CCU8 CAPCOM units

The global start command enables timers to be started, independently of the CAPCOM unit they belong to. The global start means that the timers are synchronized and all timing can be controlled in parallel, with many different kinds of generated output patterns.

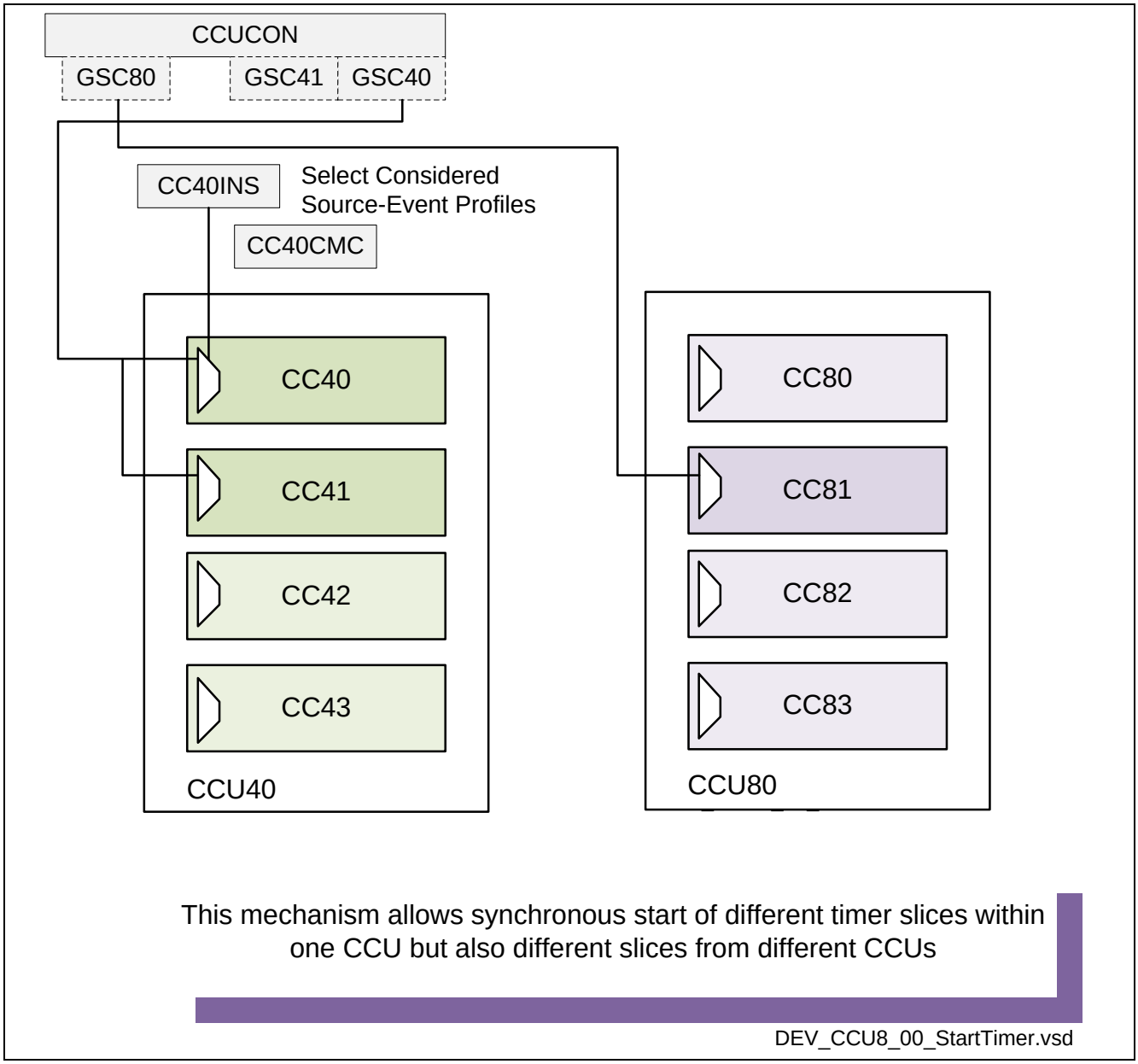


Figure 11 External event control with global start command

2 Dynamic control of timer functions on external events

2.1 Introduction

The external events control distribution to CCUs (including CCU8) allows for advanced applications with synchronized timer control. For example, in motor drive and power control such as 3-level inverters requiring 12 synchronized PWMs.

2.1.1 External control basics

A slice can have its input functions controlled by external sources. The external source(s), active mode(s) and input function(s) should be mapped to the 3 inputs of the slice in the CC8yINS and CC8yCMC registers. Alternatives to extend function modes can be added by selections in the CC8yTC timer slice control register.

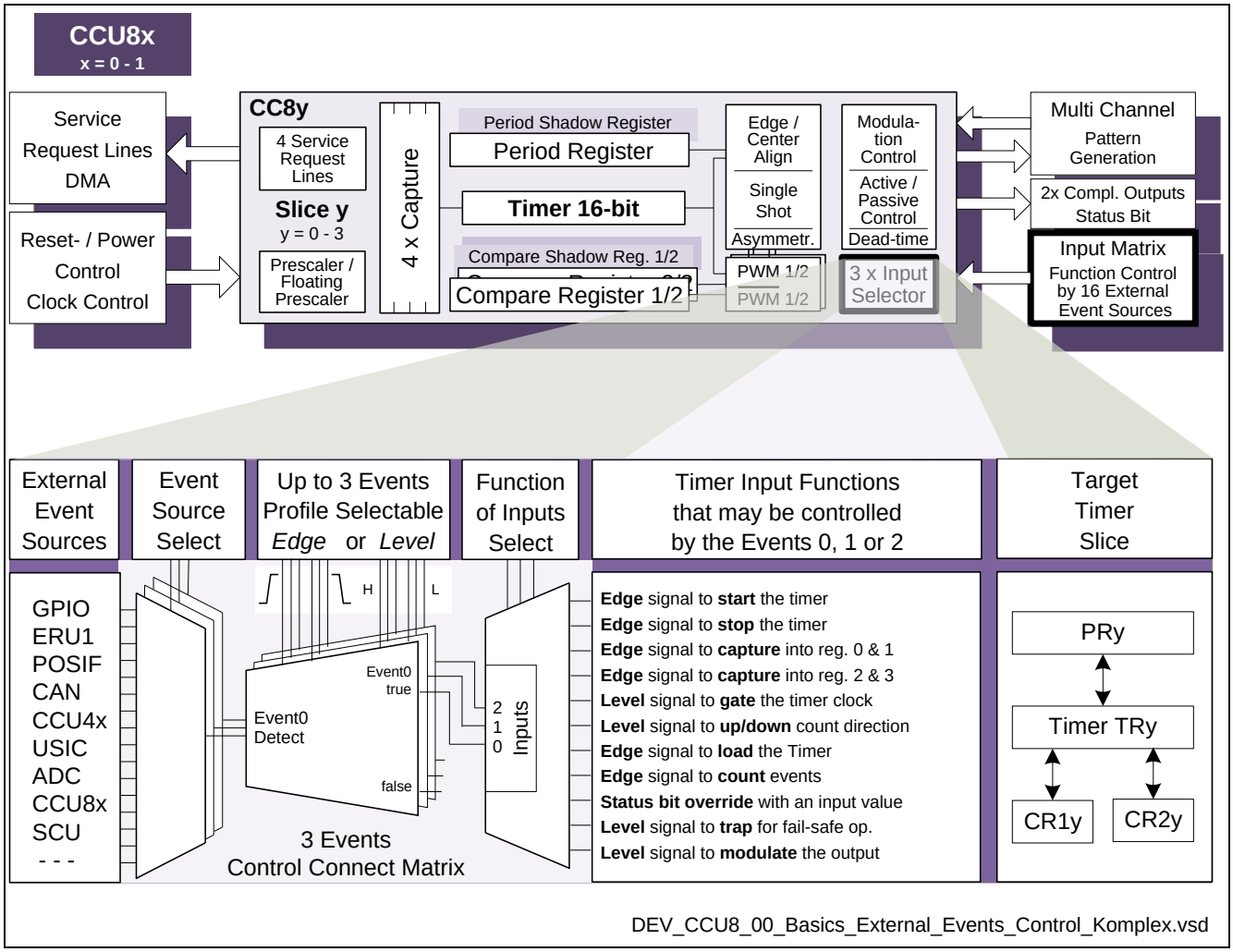


Figure 12 Timer slice input functions control on external events via the system interconnect matrix

An external event control request can be an edge or level event signal from a peripheral unit or a GPIO. It can be linked to the CCU8xCC8y slice's input selection stages via a comprehensive matrix. A slice with any of its 3 events setup detects a considered source-event-input profile and can be function controlled "remotely" this way.

2.1.2 Selection of external events control sources

CCU8xCC8y input functions can be linked to external trigger requests from sources such as: GPIO, ERU, POSIF, CAN, CCU4x, USIC, ADC, CCU8x or SCU. Pin connections are given by the top-level interconnect matrix and the CC8yINS[P:A] input select vector - and function select by the CC8yCMC register.

A CC8y internal event is also regarded as an external event. This means a CC8y can control itself by its own events.

2.1.3 Selection of external events control of input functions

There are 11 timer input functions (such as start the timer), controllable by external events via 3 selectable input lines with configurable source-event profile conditions to the timer slices CC8y (y=0-3) of a CCU8x unit for start, stop, capture0-3, gate, up/down, load, count, bit override, trap and modulate output control.

The input functions are, due to their nature, controlled by either event edge or event level signals.

2.1.4 Extended slice input functions

There are some extended input functions in the CC8yTC register, for the options flush/start, flush/stop or just flush the timer and for an extended capture mode option that via a read access register (ECRD) setup simplifies administration of capture registers and full-flags, when more than one slice is used in capture mode.

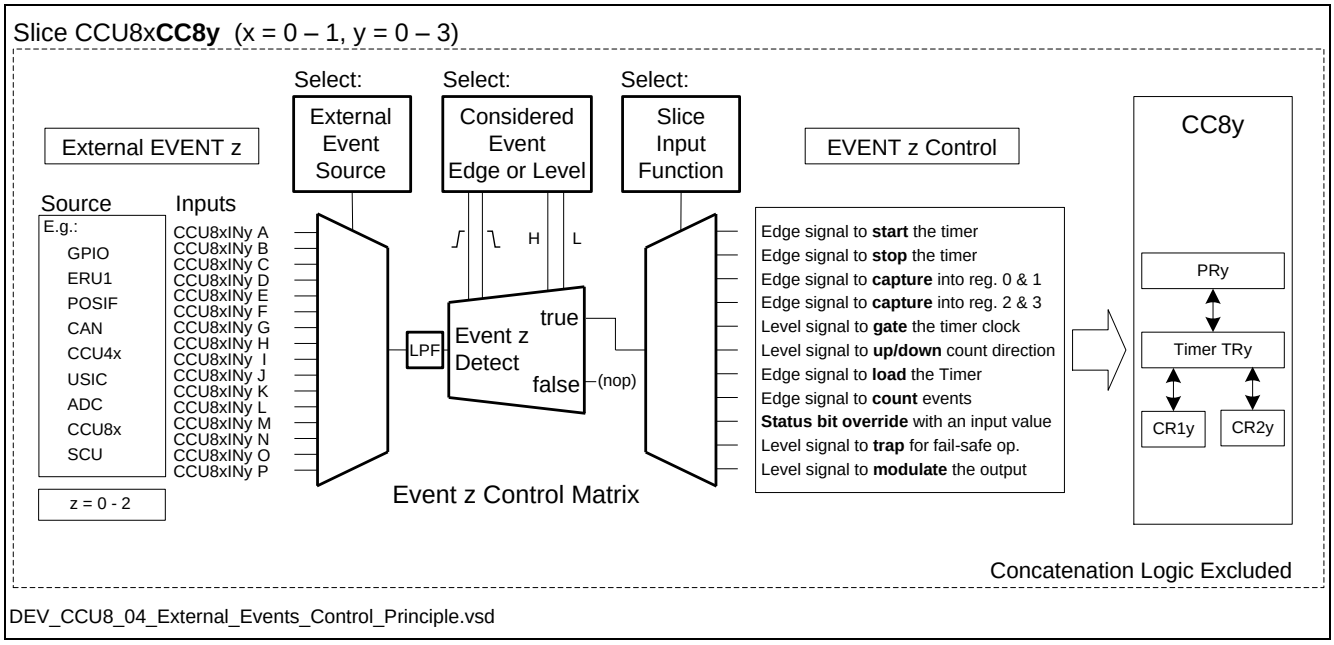


Figure 13 Principal block diagram illustrating external event control of a CCU8y timer slice

2.1.5 External control by timer events

A timer event can either trigger external actions via the top-level interconnect matrix or request for an interrupt. Each CAPCOM8 has four service request lines and each slice has a dedicated output signal CC8ySR[3...0] selectable to a line via CC8ySRS. This means timer slice events can request for direct peripheral actions or request an interrupt.

2.1.6 Top-level control of event request to/from a timer slice

Top-level control also means conditional control of event requests between a slice and other action providers. The Event Request Unit (ERU1) together with the top-level interconnect matrix can combine, control and link event signals according to user defined request-to-action event patterns, such as invoke I/O states, time windowing etc.

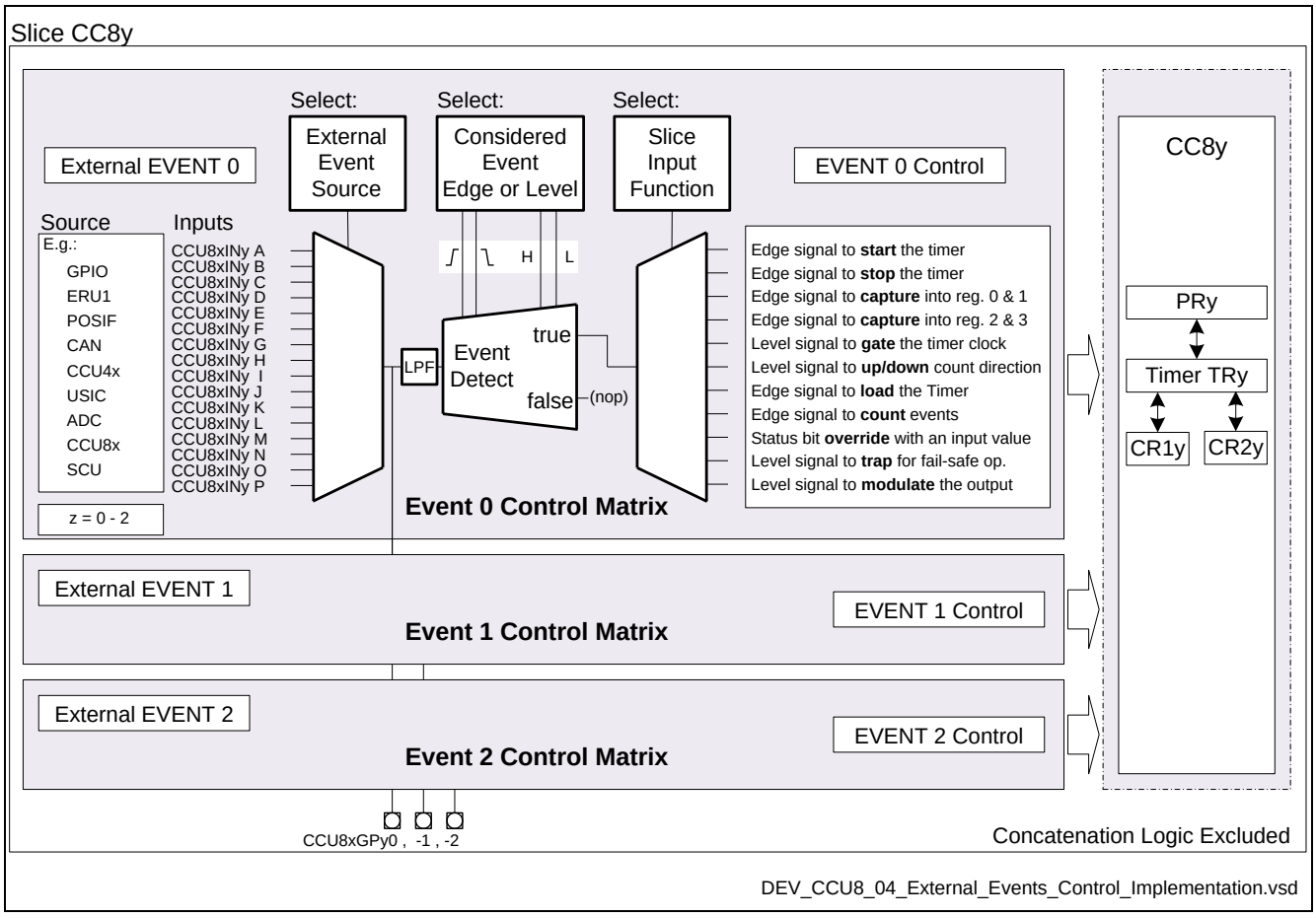


Figure 14 Block diagram of the external event control implementation

2.2 Example application: triggering an ADC conversion to change CCU8 duty cycle

In this example, the CCU80.80 slice is configured in edge-aligned mode with a frequency of 24 kHz and a 50% duty cycle on both channels 1 and 2. Each compare match event on channel 1 triggers an ADC queue conversion. An ADC channel event is triggered if the conversion result is within the set boundary limits (Upper boundary = 4000, Lower boundary = 1000). In the ADC channel event, the ADC conversion result is saved and a software variable, ADC_INBOUND, is set. This is used as a marker that an ADC conversion has occurred. During a period match event, if ADC_INBOUND is set, the duty cycle on channel 2 is updated. This example is based on the XMC4500.

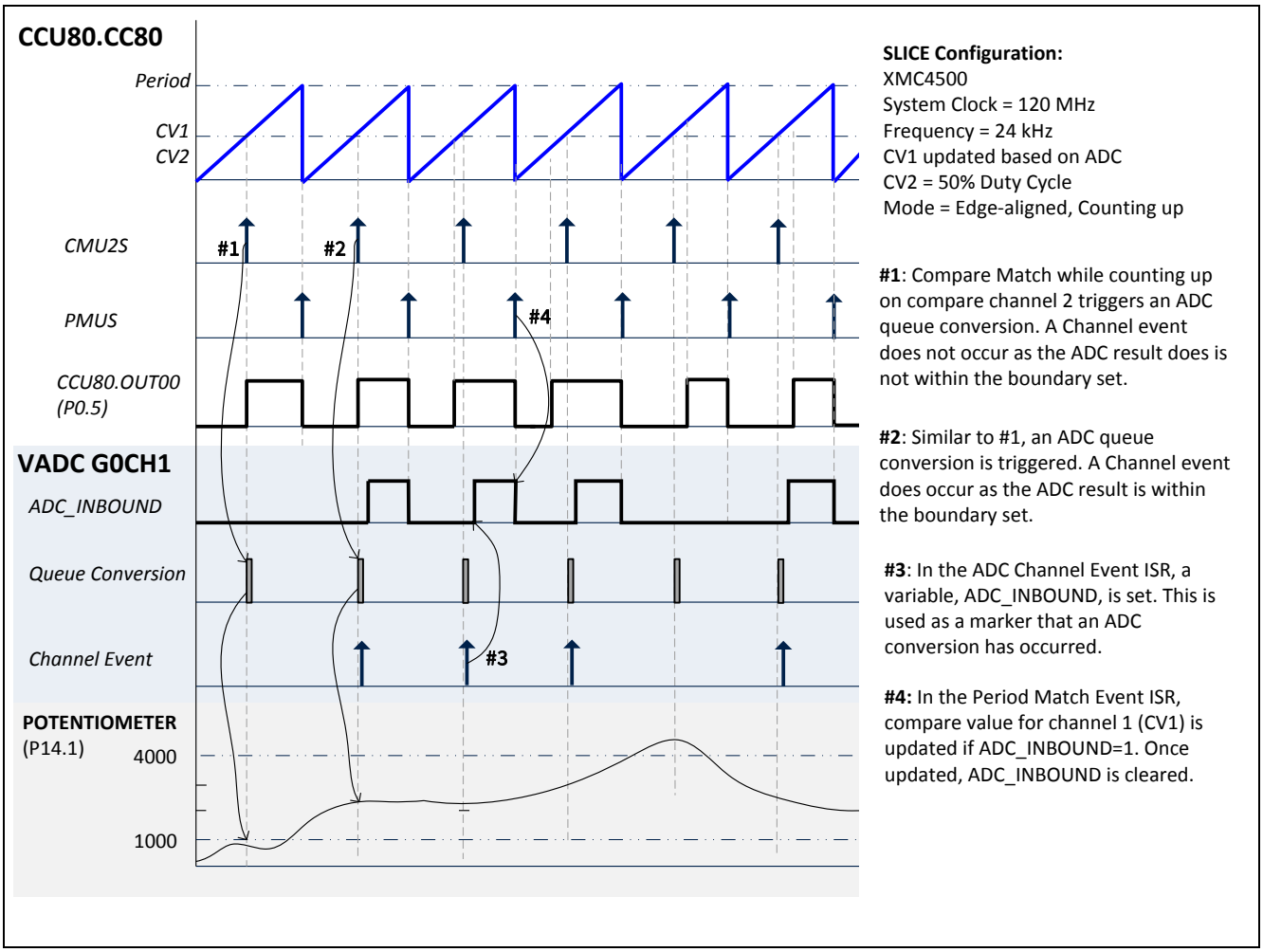


Figure 15 Example: triggering an ADC conversion to change CCU8 duty cycle

2.2.2 Deriving the period and compare values

The clock relationship between f_{PWM}, f_{tclk} and f_{ccu8} is calculated as shown below:

- f_{ccu8} is the frequency of the CCU8 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 this example, the default prescaler factor 0 is used. This results in a prescaler value of 1 and a timer resolution of 8.33 nS.
- In order for, f_{PWM} (frequency of the PWM signal) to be 24 kHz, the CCU8_CC80.PRS register is loaded with the value 4999.

Timer frequency: $f_{tclk} = \frac{f_{ccu8}}{Prescaler}$

Period value: $CCU8_{CC80}.PRS = \frac{f_{tclk}}{f_{PWM}} - 1$

Compare value: $CCU8_{CC80}.CRS = (1 - DC) * (PRS + 1)$

Table 1 Calculated prescaler factor, period and compare values

Type	Calculated value
Prescaler value	$2^0 = 0$
Period @1Hz frequency	4999
Compare value @50% DC (At initialization, CV1 = CV2)	2500

2.2.3 Macro and variable settings

XMC™ Lib project includes:

```
#include <xmc_ccu8.h>
#include <xmc_gpio.h>
#include <xmc_scu.h>
#include <xmc_vadc.h>
```

Project macro definitions for CCU8:

```
#define MODULE_PTR              CCU80
#define MODULE_NUMBER           (0U)
#define SLICE0_PTR              CCU80_CC80
#define SLICE0_NUMBER           (0U)
#define SLICE0_OUTPUT00         P0_5
```

Project macro definitions for ADC:

```
#define RES_REG_NUMBER          (0)
#define CHANNEL_NUMBER          (1U)
#define VADC_GROUP_PTR          (VADC_G0) /* P14.1 */
#define VADC_GROUP_ID           (0)
#define IRQ_PRIORITY            (10U)
```

Project variables definition:

```
volatile uint16_t CURRENT_PWM;
volatile bool ADC_INBOUND = 1;
```

2.2.4 XMC™ Lib peripheral configuration structure

XMC™ System Clock Unit (SCU) configuration:

```
/* XMC Clock configuration structure */
XMC_SCU_CLOCK_CONFIG_t clock_config = {
  .syspll_config.n_div = 80U,
  .syspll_config.p_div = 2U,
  .syspll_config.k_div = 4U,
  .syspll_config.mode = XMC_SCU_CLOCK_SYSPLL_MODE_NORMAL,
  .syspll_config.clksrc = XMC_SCU_CLOCK_SYSPLLCLKSRC_OSCHP,
  .enable_oschp = true,
  .enable_osculp = false,
  .calibration_mode = XMC_SCU_CLOCK_FOFI_CALIBRATION_MODE_FACTORY,
  .fstdby_clksrc = XMC_SCU_HIB_STDBYCLKSRC_OSI,
  .fsys_clksrc = XMC_SCU_CLOCK_SYSCLKSRC_PLL,
  .fsys_clkdiv = 1U,
  .fcpu_clkdiv = 1U,
  .fccu_clkdiv = 1U,
  .fperipheral_clkdiv = 1U
};
```

XMC™ Capture/Compare Unit 8 (CCU8) configuration for SLICE0:

```
XMC_CCU8_SLICE_COMPARE_CONFIG_t SLICE_config =
```

Dynamic control of timer functions on external events

```
{
  .timer_mode           = (uint32_t) XMC_CCU8_SLICE_TIMER_COUNT_MODE_EA,
  .monoshot             = (uint32_t) false,
  .shadow_xfer_clear    = (uint32_t) 0U,
  .dither_timer_period  = (uint32_t) 0U,
  .dither_duty_cycle    = (uint32_t) 0U,
  .mcm_ch1_enable       = (uint32_t) false,
  .mcm_ch2_enable       = (uint32_t) false,
  .slice_status         = (uint32_t) XMC_CCU8_SLICE_STATUS_CHANNEL_1,
  .prescaler_mode       = (uint32_t) XMC_CCU8_SLICE_PRESCALER_MODE_NORMAL,
  .passive_level_out0   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out1   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out2   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out3   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .asymmetric_pwm       = (uint32_t) 0U,
  .invert_out0          = (uint32_t) 0U,
  .invert_out1          = (uint32_t) 1U,
  .invert_out2          = (uint32_t) 0U,
  .invert_out3          = (uint32_t) 1U,
  .prescaler_initval    = (uint32_t) 0U,
  .float_limit          = (uint32_t) 0U,
  .dither_limit         = (uint32_t) 0U,
  .timer_concatenation  = (uint32_t) 0U
};
```

XMC™ GPIO configuration:

```
// Configuration for A2 class pads: Port0.5
XMC_GPIO_CONFIG_t OUTPUT_strong_sharp_config =
{
       .mode            = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3,
       .output_level    = XMC_GPIO_OUTPUT_LEVEL_LOW,
       .output_strength = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
};
```

XMC™ VADC configuration:

```
/* Initialization data of VADC Global resources */
XMC_VADC_GLOBAL_CONFIG_t g_global_handle =
{
 .disable_sleep_mode_control = false,
 .clock_config  = {
                .analog_clock_divider   = 3U,
                .msb_conversion_clock   = 0U,
                .arbiter_clock_divider  = 1U
                },
 .class0        = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                },
 .class1        = {
```

```
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                },
 .data_reduction_control = 0,
 .wait_for_read_mode     = true,
 .event_gen_enable       = false,
 .boundary0              = 0,
 .boundary1              = 0
  };

/* Initialization data of a VADC group */
XMC_VADC_GROUP_CONFIG_t g_group_handle =
{
 .class0        = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                },
 .class1        = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                },
 .arbitration_round_length        = 0x0U,
 .arbiter_mode                    = XMC_VADC_GROUP_ARBMODE_ALWAYS,
 .boundary0                       = 1000U, /* Boundary-0 */
 .boundary1                       = 4000U, /* Boundary-1 */
 .emux_config   = {
                .emux_mode                = XMC_VADC_GROUP_EMUXMODE_SWCTRL,
                .stce_usage               = 0,
                .emux_coding              = XMC_VADC_GROUP_EMUXCODE_BINARY,
                .starting_external_channel = 0,
                .connected_channel        = 0
                }
};

/* Identifier of the hardware group */
XMC_VADC_GROUP_t *g_group_identifier =VADC_GROUP_PTR;

/* Channel configuration data */
XMC_VADC_CHANNEL_CONFIG_t g_channel_handle =
{
  .channel_priority            = 1U,
  .input_class                 = XMC_VADC_CHANNEL_CONV_GROUP_CLASS1,
  .lower_boundary_select       = XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOUND0,
  .upper_boundary_select       = XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOUND1,
  .alias_channel               = (uint8_t)-1,
  .bfl                         = 0,
```

```
  .event_gen_criteria          = XMC_VADC_CHANNEL_EVGEN_INBOUND,
  .alternate_reference         = XMC_VADC_CHANNEL_REF_INTREF,
  .result_reg_number           = (uint8_t) RES_REG_NUMBER,
  .sync_conversion             = false,             /* Sync Feature disabled*/
  .result_alignment            = XMC_VADC_RESULT_ALIGN_RIGHT,
  .use_global_result           = false,
  .broken_wire_detect_channel  = false,
  .broken_wire_detect          = false
};

/* Result configuration data */
XMC_VADC_RESULT_CONFIG_t g_result_handle = {
        .post_processing_mode   = XMC_VADC_DMM_REDUCTION_MODE,
        .data_reduction_control = 0,
        .part_of_fifo           = false, /* No FIFO */
        .wait_for_read_mode     = false, /* WFS */
        .event_gen_enable       = false  /* No result event */
};

/* Queue hardware configuration data */
XMC_VADC_QUEUE_CONFIG_t g_queue_handle =
{
  .req_src_priority = (uint8_t)3U, /* Highest Priority = 3, Lowest = 0 */
  .conv_start_mode  = XMC_VADC_STARTMODE_WFS,
  .external_trigger = (bool) true, /* External trigger enabled*/
  .trigger_signal   = XMC_CCU_80_SR2,
  .trigger_edge     = XMC_VADC_TRIGGER_EDGE_RISING,
  .gate_signal      = XMC_VADC_REQ_GT_A,
  .timer_mode       = (bool) false, /* No timer mode */
};

/* Queue Entry */
XMC_VADC_QUEUE_ENTRY_t g_queue_entry =
{
 .channel_num           = CHANNEL_NUMBER,
 .refill_needed         = true, /* Refill is needed */
 .generate_interrupt    = false, /* Interrupt generation is needed */
 .external_trigger      = true /* External trigger is required */
};
```

2.2.5 Interrupt service routine function implementation

The CCU80 interrupt handler function to update the duty cycle on channel 1 at every period match event:

```
/* Interrupt handler - Period Match Interrupt; Updates the PWM frequency as long as ADC
conversion within boundary limits set */
void CCU80_0_IRQHandler(void)
{
        /* Acknowledge Period Match event*/
        XMC_CCU8_SLICE_ClearEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
```

```
        /* Set up new PWM value */
        if(ADC_INBOUND==1)
        {
                XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
                        XMC_CCU8_SLICE_COMPARE_CHANNEL_1, CURRENT_PWM);

                XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_0);
                ADC_INBOUND=0;
        }
}
```

The VADC interrupt handler function generates a channel event when the value is within the boundary limit defined:

```
/* Interrupt handler - Channel Interrupt; this is entered if the boundary set is in
boundary limits set*/
void VADC0_G0_0_IRQHandler(void)
{
        XMC_VADC_RESULT_SIZE_t result;

        /* Read the result register */
        result = XMC_VADC_GROUP_GetResult(g_group_identifier,RES_REG_NUMBER);

        /* Clear result event */
        XMC_VADC_GROUP_ChannelClearEvent(g_group_identifier,CHANNEL_NUMBER);

        /* Set marker for PWM duty cycle update*/
        if(ADC_INBOUND == 0 )
        {
                CURRENT_PWM = result;
                ADC_INBOUND = 1;
        }
}
```

2.2.6 Main function implementation

Before the start and execution of timer slice software for the first time, the CCU8 must be initialized appropriately using the following sequence:

- Clock setup

```
        /* Ensure clock frequency is set at 120 MHz */
        XMC_SCU_CLOCK_Init(&clock_config);
```

- Enable clock, enable prescaler block and configure global control:

```
        /* Enable CCU8 module */
        XMC_CCU8_Init(MODULE_PTR, XMC_CCU8_SLICE_MCMS_ACTION_TRANSFER_PR_CR);

        /* Start the prescaler */
        XMC_CCU8_StartPrescaler(MODULE_PTR);

        /* Ensure fCCU reaches CCU80 */
        XMC_CCU8_SetModuleClock(MODULE_PTR, XMC_CCU8_CLOCK_SCU);
```

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

```
/* Configure CCU8x_CC8y slice as timer */
XMC_CCU8_SLICE_CompareInit(SLICE0_PTR, &SLICE_config);

/* Set period match value of the timer */
XMC_CCU8_SLICE_SetTimerPeriodMatch(SLICE0_PTR, 4999U);

/* Set timer compare match value for channel 1 - 50% duty */
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
        XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 2500U);

/* Set timer compare match value for channel 2 - 50% duty */
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
        XMC_CCU8_SLICE_COMPARE_CHANNEL_2, 2500U);

/* Transfer value from shadow timer registers to actual timer registers */
XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_0);

/* Configure events */
/* Enable events: Period Match and Compare Match-Ch2 */
XMC_CCU8_SLICE_EnableEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);

XMC_CCU8_SLICE_EnableEvent(SLICE0_PTR, \
        XMC_CCU8_SLICE_IRQ_ID_COMPARE_MATCH_UP_CH_2);

/* Connect event to SR0 and SR2 */
XMC_CCU8_SLICE_SetInterruptNode(SLICE0_PTR, \
        XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH, XMC_CCU8_SLICE_SR_ID_0);

XMC_CCU8_SLICE_SetInterruptNode(SLICE0_PTR,\
        XMC_CCU8_SLICE_IRQ_ID_COMPARE_MATCH_UP_CH_2, XMC_CCU8_SLICE_SR_ID_2);

/* Configure NVIC */
/* Set priority */
NVIC_SetPriority(CCU80_0_IRQn, 63U);

/* Enable IRQ */
NVIC_EnableIRQ(CCU80_0_IRQn);

/*Initializes the GPIO*/
XMC_GPIO_Init(SLICE0_OUTPUT00, &OUTPUT_strong_sharp_config);
```

- Configure ADC queue settings:

```
/* Initialize the VADC global registers */
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);

/* Configure a conversion kernel */
XMC_VADC_GROUP_Init(g_group_identifier, &g_group_handle);

/* Configure the queue request source of the aforesaid conversion kernel */
```

```
XMC_VADC_GROUP_QueueInit(g_group_identifier, &g_queue_handle);

/* Configure a channel belonging to the aforesaid conversion kernel */
XMC_VADC_GROUP_ChannelInit(g_group_identifier,CHANNEL_NUMBER, &g_channel_handle);

/* Configure a result resource belonging to the aforesaid conversion kernel */
XMC_VADC_GROUP_ResultInit(g_group_identifier, RES_REG_NUMBER, &g_result_handle);

/* Set priority of NVIC node meant to be connected to Kernel Request source event*/
NVIC_SetPriority(VADC0_G0_0_IRQn, IRQ_PRIORITY);

/* Connect RS Event to the NVIC nodes */
XMC_VADC_GROUP_ChannelSetEventInterruptNode \
        (g_group_identifier, CHANNEL_NUMBER, XMC_VADC_SR_GROUP_SR0);

/* Enable IRQ */
NVIC_EnableIRQ(VADC0_G0_0_IRQn);

/* Enable the analog converters */
XMC_VADC_GROUP_SetPowerMode(g_group_identifier, XMC_VADC_GROUP_POWERMODE_NORMAL);

/* Perform calibration of the converter */
XMC_VADC_GLOBAL_StartupCalibration(VADC);

/* Add the channel to the queue */
XMC_VADC_GROUP_QueueInsertChannel(g_group_identifier, g_queue_entry);
```

- Start timer running:

```
/* Get the slice out of idle mode */
XMC_CCU8_EnableClock(MODULE_PTR, SLICE0_NUMBER);

/* Start the timer */
XMC_CCU8_SLICE_StartTimer(SLICE0_PTR);
```

2.3 Example application: generating a CCU8 TRAP with ADC fast compare

All applications are defined with a set of operating conditions so that they function normally. The usual way to achieve this is to monitor certain signals (for example, input voltages, feedback current) to ensure that the application is functioning within the boundary conditions set.

In this example, based on the XMC4500, we are using the VADC fast compare mode to monitor a signal input voltage to ensure that it does not exceed the upper boundary limit (of 4000) that has been set. Once this happens, a boundary flag is set. The boundary flag is used as an input for an external trap event. Once the external trap event is triggered, the output signals (CCU80.OUT00 and CCU80.OUT02) are set to a passive output state. The trap exit condition selected allows the trap to be exited automatically by hardware once the signal input voltage is within the boundary again.

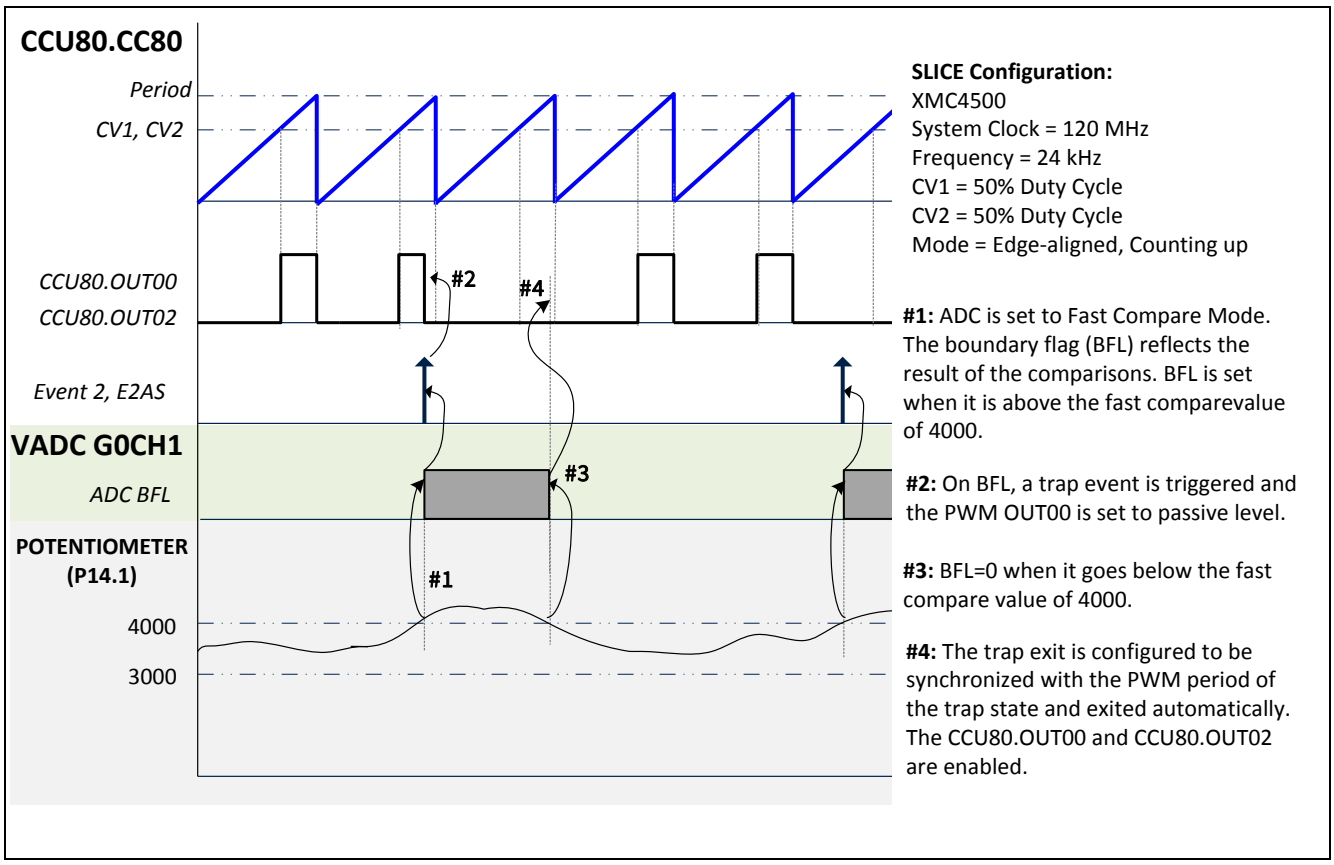


Figure 16 Example: generating a CCU8 trap with ADC fast compare

2.3.1 Theory of operation

With the limit-checking feature of VADC on the XMC™ series, every digital conversion result can be automatically compared to an upper and a lower boundary value. A channel event can be generated when the result of a conversion is inside or outside of a user-defined band, enabling a service request to only be issued under certain pre-defined conditions (depending on the boundary definition). This feature supports automatic range monitoring and minimizes the CPU load by issuing CCU8 TRAP service requests only under certain predefined conditions.

The boundary flags exist to monitor if a value has crossed the activation boundary. These flags can be represented as a change in the bitfield BFLy of the Boundary Flag Register (GxBFL), and can act as a trigger signal for the CCU8 TRAP to protect the hardware.

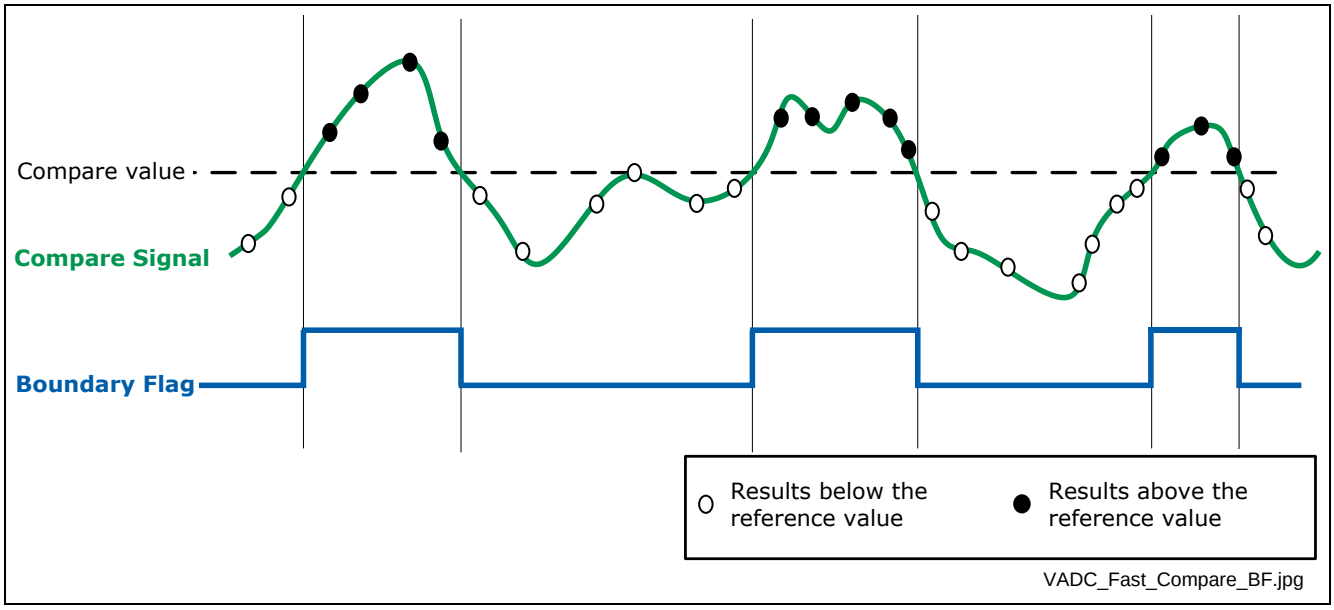


Figure 17 Boundary flag in fast compare mode

The TRAP functionality allows the PWM outputs to react on the state of an input pin. This functionality can be used to switch off the power devices if the TRAP input becomes active. When a TRAP condition is detected at the selected input pin, both the trap flag and the trap state bit are set to 1B. The trap state is entered immediately by setting the CCU8xOUTy into the programmed PASSIVE state.

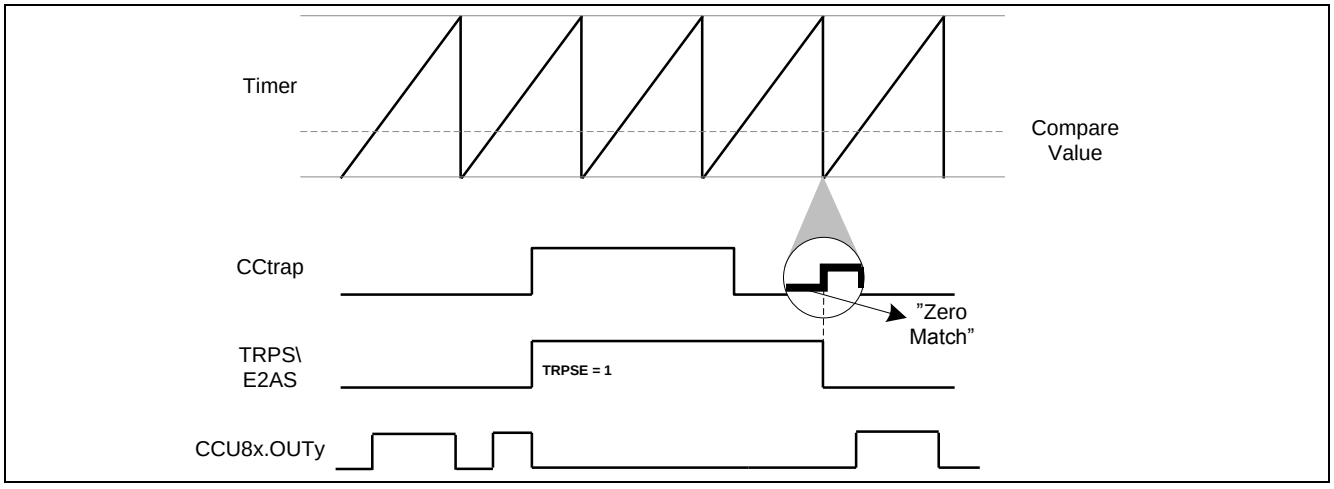


Figure 18 Trap synchronization with PWM signal

It is also possible to synchronize the exiting of the TRAP state with the PWM signal as shown in Figure 18. This function is enabled when the bitfield CC8yTC.TRPSE = 1B.

2.3.2 Deriving the period and compare values

The clock relationship between f_{PWM}, f_{tclk} and f_{ccu8} is calculated as shown below:

- $f_{ccu8}$is the frequency of the CCU8 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 this example, the default prescaler factor 0. This results in a prescaler value of 1 and a timer resolution of 8.33 ns.

- In order for, f_{PWM} (frequency of the PWM signal) to be 24 kHz, the CCU8_CC80.PRS register is loaded with the value 4999.

Timer frequency: $f_{tclk} = \frac{f_{ccu8}}{Prescaler}$

Period value: $CCU8_{CC80}.PRS = \frac{f_{tclk}}{f_{PWM}} - 1$

Compare value: $CCU8_{CC80}.CRS = (1 - DC) * (PRS + 1)$

Table 2 Calculated prescaler factor, period and compare values

Type	Calculated value
Prescaler factor	$2^0 = 0$
Period @24 kHz frequency	4999
Compare value @50% DC (At initialization, CV1 = CV2)	2500

2.3.3 Macro and variable settings

XMC™ Lib project includes:

```
#include <xmc_ccu8.h>
#include <xmc_gpio.h>
#include <xmc_scu.h>
#include <xmc_vadc.h>
```

Project macro definitions for CCU8:

```
#define MODULE_PTR              CCU80
#define MODULE_NUMBER           (0U)
#define SLICE0_PTR              CCU80_CC80
#define SLICE0_NUMBER           (0U)
#define SLICE0_OUTPUT00         P0_5
#define SLICE0_OUTPUT02         P0_10
```

Project macro definitions for ADC:

```
#define RES_REG_NUMBER          (0)
#define CHANNEL_NUMBER          (1U)
#define VADC_GROUP_PTR          (VADC_G0) /* P14.1 */
#define VADC_GROUP_ID           (0)
#define IRQ_PRIORITY            (10U)
#define FAST_COMPARE_VAL        (4000U)
```

2.3.4 XMC™ Lib peripheral configuration structure

XMC™ System Clock Unit (SCU) configuration:

```
/* XMC Clock configuration structure */
XMC_SCU_CLOCK_CONFIG_t clock_config = {
  .syspll_config.n_div = 80U,
  .syspll_config.p_div = 2U,
```

Dynamic control of timer functions on external events

```
  .syspll_config.k_div = 4U,
  .syspll_config.mode = XMC_SCU_CLOCK_SYSPLL_MODE_NORMAL,
  .syspll_config.clksrc = XMC_SCU_CLOCK_SYSPLLCLKSRC_OSCHP,
  .enable_oschp = true,
  .enable_osculp = false,
  .calibration_mode = XMC_SCU_CLOCK_FOFI_CALIBRATION_MODE_FACTORY,
  .fstdby_clksrc = XMC_SCU_HIB_STDBYCLKSRC_OSI,
  .fsys_clksrc = XMC_SCU_CLOCK_SYSCLKSRC_PLL,
  .fsys_clkdiv = 1U,
  .fcpu_clkdiv = 1U,
  .fccu_clkdiv = 1U,
  .fperipheral_clkdiv = 1U
};
```

XMC™ Capture/Compare Unit 8 (CCU8) Configuration for SLICE0:

```
XMC_CCU8_SLICE_COMPARE_CONFIG_t SLICE_config =
{
  .timer_mode           = (uint32_t) XMC_CCU8_SLICE_TIMER_COUNT_MODE_EA,
  .monoshot             = (uint32_t) false,
  .shadow_xfer_clear    = (uint32_t) 0U,
  .dither_timer_period  = (uint32_t) 0U,
  .dither_duty_cycle    = (uint32_t) 0U,
  .mcm_ch1_enable       = (uint32_t) false,
  .mcm_ch2_enable       = (uint32_t) false,
  .slice_status         = (uint32_t) XMC_CCU8_SLICE_STATUS_CHANNEL_1,
  .prescaler_mode       = (uint32_t) XMC_CCU8_SLICE_PRESCALER_MODE_NORMAL,
  .passive_level_out0   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out1   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out2   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .passive_level_out3   = (uint32_t) XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
  .asymmetric_pwm       = (uint32_t) 0U,
  .invert_out0          = (uint32_t) 0U,
  .invert_out1          = (uint32_t) 1U,
  .invert_out2          = (uint32_t) 0U,
  .invert_out3          = (uint32_t) 1U,
  .prescaler_initval    = (uint32_t) 0U,
  .float_limit          = (uint32_t) 0U,
  .dither_limit         = (uint32_t) 0U,
  .timer_concatenation  = (uint32_t) 0U
};

XMC_CCU8_SLICE_EVENT_CONFIG_t TRAP_config =
{
  .mapped_input = XMC_CCU8_SLICE_INPUT_I,       /* VADC.GOBFL0 */
  .edge = XMC_CCU8_SLICE_EVENT_EDGE_SENSITIVITY_NONE,
  .level = XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_HIGH,
  .duration = XMC_CCU8_SLICE_EVENT_FILTER_DISABLED
};
```

Dynamic control of timer functions on external events

XMC™ GPIO configuration:

```
// Configuration for A2 class pads: Port0.5
XMC_GPIO_CONFIG_t OUTPUT_strong_sharp_config =
{
        .mode                = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3,
        .output_level        = XMC_GPIO_OUTPUT_LEVEL_LOW,
        .output_strength     = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SHARP_EDGE
};

// Configuration for A1+ class pads: Port0.10
XMC_GPIO_CONFIG_t OUTPUT_strong_soft_config =
{
       .mode            = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3,
       .output_level    = XMC_GPIO_OUTPUT_LEVEL_LOW,
       .output_strength = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
};
```

XMC™ VADC configuration:

```
/* Initialization data of VADC Global resources */
XMC_VADC_GLOBAL_CONFIG_t g_global_handle =
{
 .disable_sleep_mode_control = false,
 .clock_config = {
                .analog_clock_divider   = 3U,
                .msb_conversion_clock   = 0U,
                .arbiter_clock_divider  = 1U
                 },
 .class0 = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                        },
 .class1 = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                        },
 .data_reduction_control = 0U,
 .wait_for_read_mode     = true,
 .event_gen_enable       = false,
 .boundary0              = 0U,
 .boundary1              = 0U
  };

/* Initialization data of a VADC group */
XMC_VADC_GROUP_CONFIG_t g_group_handle =
{
// .group_num = VADC_GROUP_ID,
 .class0 = {
```

```
                .conversion_mode_standard       = XMC_VADC_CONVMODE_12BIT,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                        },
 .class1 = {
                .conversion_mode_standard       = XMC_VADC_CONVMODE_FASTCOMPARE,
                .sample_time_std_conv           = 3U,
                .conversion_mode_emux           = XMC_VADC_CONVMODE_12BIT,
                .sampling_phase_emux_channel    = 3U
                        },
 .arbitration_round_length        = 0x0U,
 .arbiter_mode                    = XMC_VADC_GROUP_ARBMODE_ALWAYS,
 .boundary0                       = 1000U, /* Boundary-0 */
 .boundary1                       = 4000U, /* Boundary-1 */
 .emux_config = {
                .emux_mode                = XMC_VADC_GROUP_EMUXMODE_SWCTRL,
                .stce_usage               = 0U,
                .emux_coding              = XMC_VADC_GROUP_EMUXCODE_BINARY,
                .starting_external_channel = 0U,
                .connected_channel        = 0U
                }
};

/* Identifier of the hardware group */
XMC_VADC_GROUP_t *g_group_identifier =VADC_GROUP_PTR;

/* Channel configuration data */
XMC_VADC_CHANNEL_CONFIG_t g_channel_handle =
{
  .channel_priority             = 1U,
  .input_class                  = XMC_VADC_CHANNEL_CONV_GROUP_CLASS1,
  .lower_boundary_select        = XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOUND0,
  .upper_boundary_select        = XMC_VADC_CHANNEL_BOUNDARY_GROUP_BOUND1,
  .alias_channel                = (uint8_t)-1,
  .boundary_flag_output_ch0     = 1,
  .event_gen_criteria           = XMC_VADC_CHANNEL_EVGEN_COMPHIGH,
  .alternate_reference          = XMC_VADC_CHANNEL_REF_INTREF,
  .result_reg_number            = (uint8_t) RES_REG_NUMBER,
  .sync_conversion              = false,                        /* Sync Feature disabled*/
  .result_alignment             = XMC_VADC_RESULT_ALIGN_RIGHT,
  .use_global_result            = false,
  .broken_wire_detect_channel   = false,
  .broken_wire_detect           = false
};

/* Result configuration data */
XMC_VADC_RESULT_CONFIG_t g_result_handle = {
        .post_processing_mode   = XMC_VADC_DMM_REDUCTION_MODE,
        .data_reduction_control = 0,
        .part_of_fifo           = false, /* No FIFO */
        .wait_for_read_mode     = false, /* WFS */
```

```
        .event_gen_enable       = false  /* No result event */
};

/* Queue hardware configuration data */
XMC_VADC_QUEUE_CONFIG_t g_queue_handle =
{
  .req_src_priority     = (uint8_t)3U, /* Highest Priority = 3, Lowest = 0 */
  .conv_start_mode      = XMC_VADC_STARTMODE_WFS,
  .external_trigger     = (bool) false, /* External trigger enabled*/
  .trigger_edge         = XMC_VADC_TRIGGER_EDGE_NONE,
  .gate_signal          = XMC_VADC_REQ_GT_A,
  .timer_mode           = (bool) false, /* No timer mode */
};

/* Queue Entry */
XMC_VADC_QUEUE_ENTRY_t g_queue_entry =
{
 .channel_num           = CHANNEL_NUMBER,
 .refill_needed         = true, /* Refill is needed */
 .generate_interrupt    = false, /* Interrupt generation is needed */
 .external_trigger      = false  /* External trigger is required */
};
```

2.3.5 Main function implementation

Before the start and execution of timer slice software for the first time, the CCU8 must be initialized appropriately using the following sequence:

- Clock setup:

```
/* Ensure clock frequency is set at 120 MHz */
XMC_SCU_CLOCK_Init(&clock_config);
```

- Enable clock, enable prescaler block and configure global control:

```
/* Enable CCU8 module */
XMC_CCU8_Init(MODULE_PTR, XMC_CCU8_SLICE_MCMS_ACTION_TRANSFER_PR_CR);

/* Get the slice out of idle mode */
XMC_CCU8_EnableClock(MODULE_PTR, SLICE0_NUMBER);

/* Start the prescaler */
XMC_CCU8_StartPrescaler(MODULE_PTR);
```

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

```
/* Configure CCU8x_CC8y slice as timer */
XMC_CCU8_SLICE_CompareInit(SLICE0_PTR, &SLICE_config);

/* Set period match value of the timer */
XMC_CCU8_SLICE_SetTimerPeriodMatch(SLICE0_PTR, 4999U);

/* Set timer compare match value for channel 1 - 50% duty */
```

```
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 2500);

/* Set timer compare match value for channel 2 - 50% duty */
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
XMC_CCU8_SLICE_COMPARE_CHANNEL_2, 2500);

/* Transfer value from shadow timer registers to actual timer registers */
XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_0);

/* Configure events */
/* Trap exit is synchronized to PWM signal*/
XMC_CCU8_SLICE_TrapConfig(SLICE0_PTR, XMC_CCU8_SLICE_TRAP_EXIT_MODE_AUTOMATIC, 1U);
XMC_CCU8_SLICE_ConfigureEvent(SLICE0_PTR,XMC_CCU8_SLICE_EVENT_2, &TRAP_config);

/* Enable events: Trap*/
XMC_CCU8_SLICE_EnableEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_EVENT2);
XMC_CCU8_SLICE_EnableTrap(SLICE0_PTR, \
        (uint32_t) (XMC_CCU8_SLICE_OUTPUT_0| \
        XMC_CCU8_SLICE_OUTPUT_2));

/*Initializes the GPIO*/
XMC_GPIO_Init(SLICE0_OUTPUT00, &OUTPUT_strong_sharp_config);
XMC_GPIO_Init(SLICE0_OUTPUT02, &OUTPUT_strong_soft_config);
```

- Configure ADC queue settings:

```
/* Initialize the VADC global registers */
XMC_VADC_GLOBAL_Init(VADC, &g_global_handle);

/* Configure a conversion kernel */
XMC_VADC_GROUP_Init(g_group_identifier, &g_group_handle);

/* Configure the queue request source of the aforesaid conversion kernel */
XMC_VADC_GROUP_QueueInit(g_group_identifier, &g_queue_handle);

/* Configure a channel belonging to the aforesaid conversion kernel */
XMC_VADC_GROUP_ChannelInit(g_group_identifier,CHANNEL_NUMBER, &g_channel_handle);

/* Configure a result resource belonging to the aforesaid conversion kernel */
XMC_VADC_GROUP_ResultInit(g_group_identifier, RES_REG_NUMBER, &g_result_handle);

/* Enable the analog converters */
XMC_VADC_GROUP_SetPowerMode(g_group_identifier, XMC_VADC_GROUP_POWERMODE_NORMAL);

/* Set Group Fast Compare value*/
XMC_VADC_GROUP_SetResultFastCompareValue(g_group_identifier, \
        RES_REG_NUMBER, (XMC_VADC_RESULT_SIZE_t)(FAST_COMPARE_VAL));

/* Perform calibration of the converter */
XMC_VADC_GLOBAL_StartupCalibration(VADC);
```

```
/* Add the channel to the queue */
XMC_VADC_GROUP_QueueInsertChannel(g_group_identifier, g_queue_entry);
```

- Start timer running:

```
/* Get the slice out of idle mode */
XMC_CCU8_EnableClock(MODULE_PTR, SLICE0_NUMBER);

/* Start the CCU8 timer */
XMC_CCU8_SLICE_StartTimer(SLICE0_PTR);
```

3 Multi phase output pattern generation

3.1 Introduction

The CAPCOM8 is a multi-purpose timer unit for signal monitoring/conditioning and Pulse Width Modulation (PWM) signal generation. It is designed with repetitive structures with multiple timer slices that have the same base functionality. The internal modularity of CCU8 translates into a software friendly system for fast code development and portability between applications.

The following image shows the main function blocks of one of the four CC8y slices on a CCU8x.

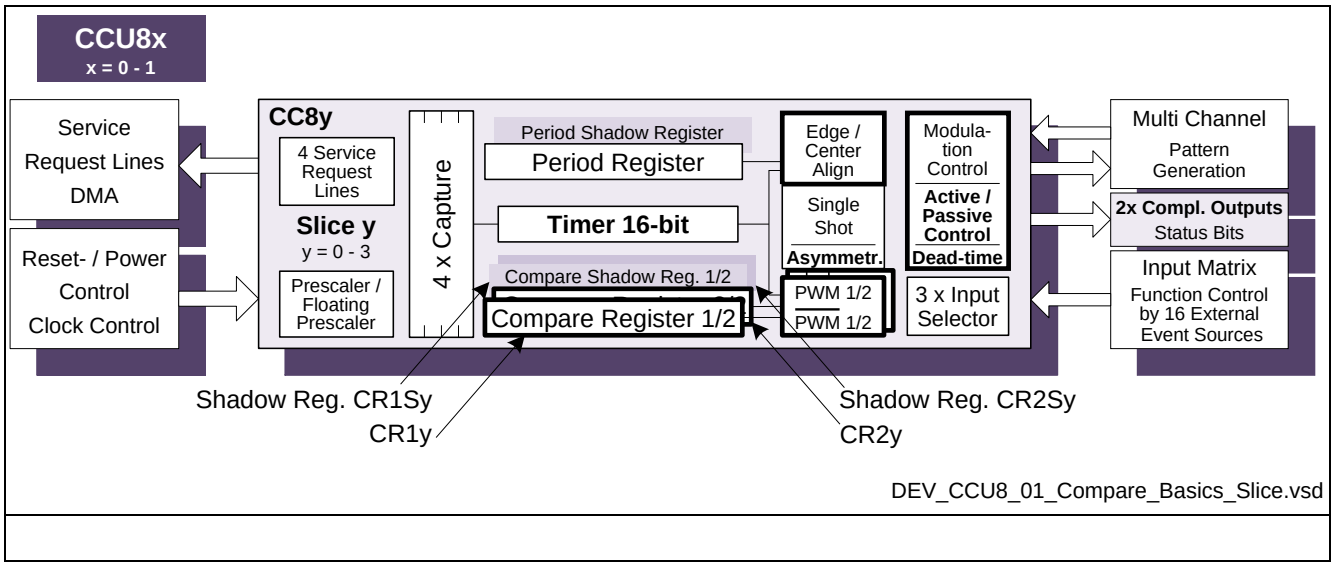


Figure 19 Timer slice compare registers and PWM related blocks

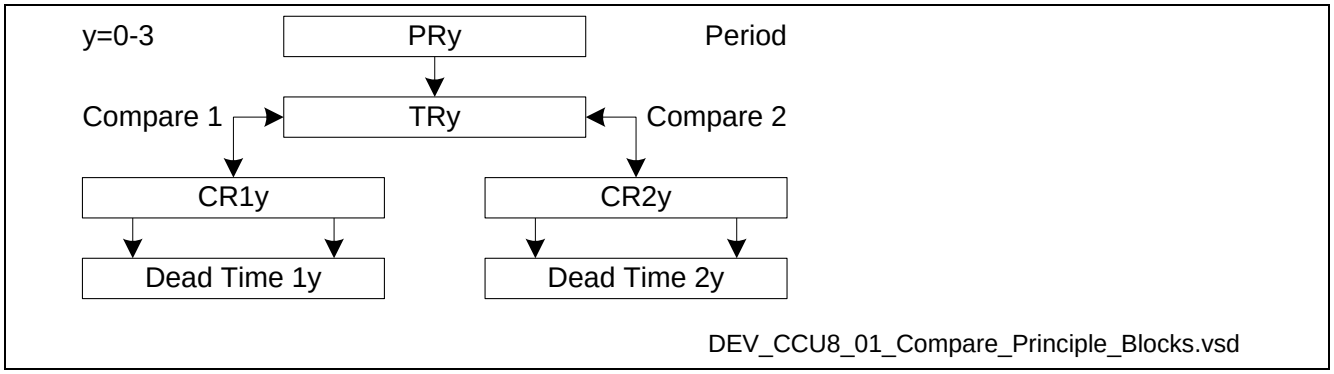


Figure 20 Two compare cChannels principle blocks

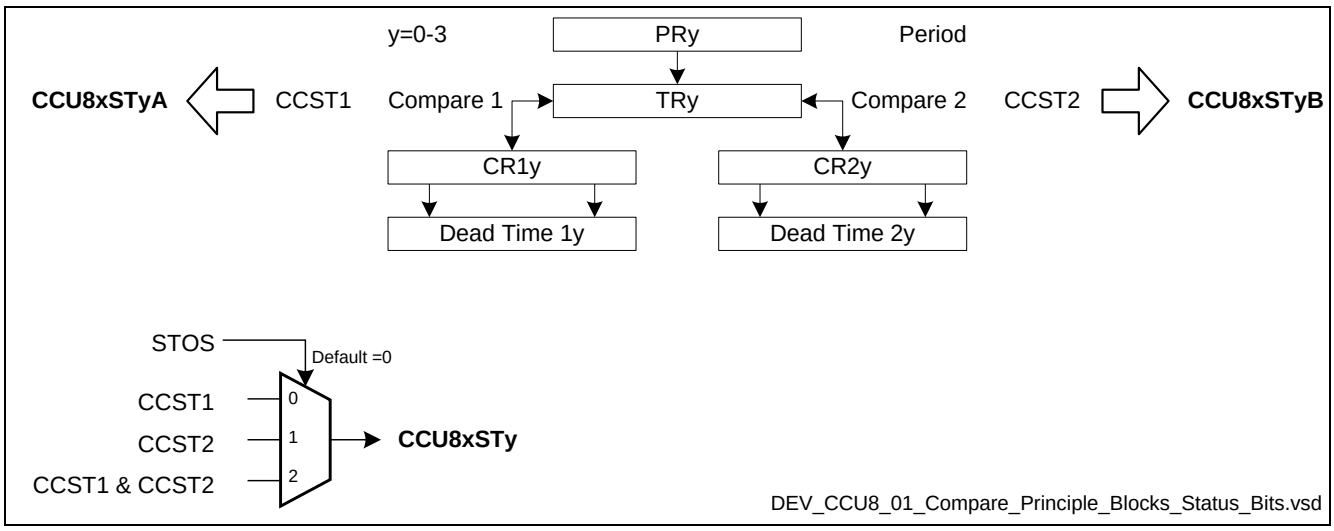


Figure 21 Two compare channels status bits

Asymmetric compare

The benefit of shadow transfers on both period-match and one-match, allows an asymmetric compare to be performed in center aligned mode by software. In addition, the CCU8x slice offers two compare registers (CC8yCR1/CR2) and the aggregated shadow registers (CC8yCR1S/CR2S). These allow asymmetric compare to be performed by hardware only.

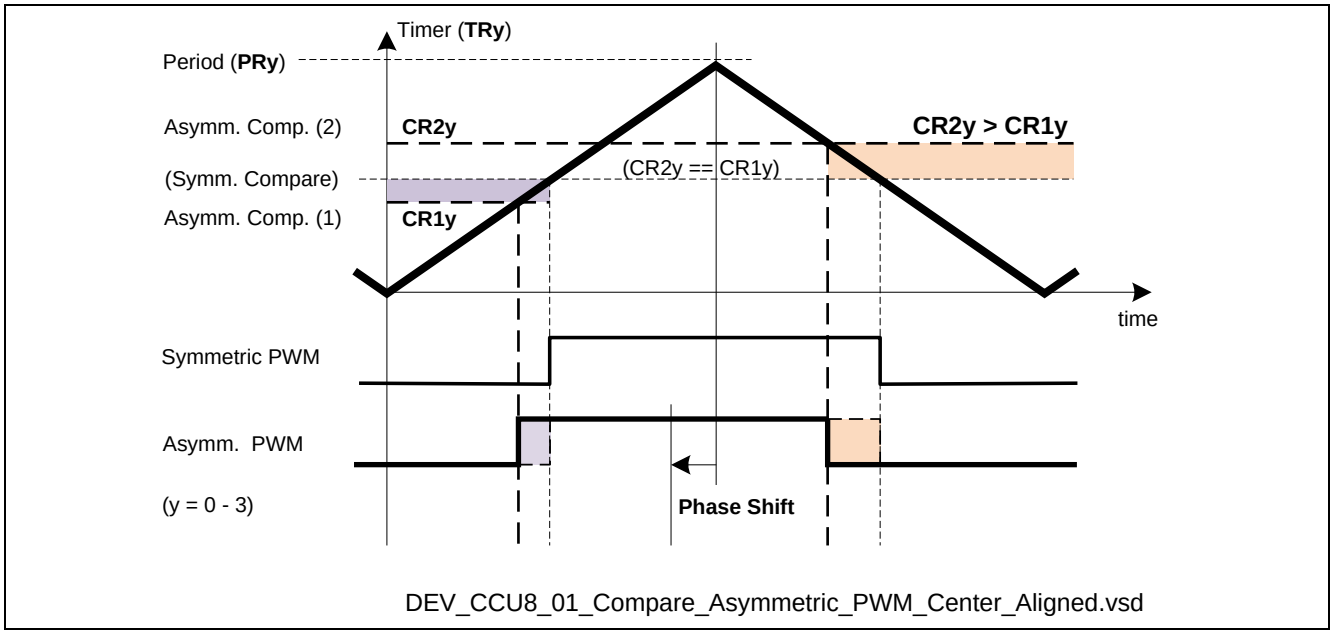


Figure 22 Symmetric PWM and asymmetric PWM (Center Aligned Mode)

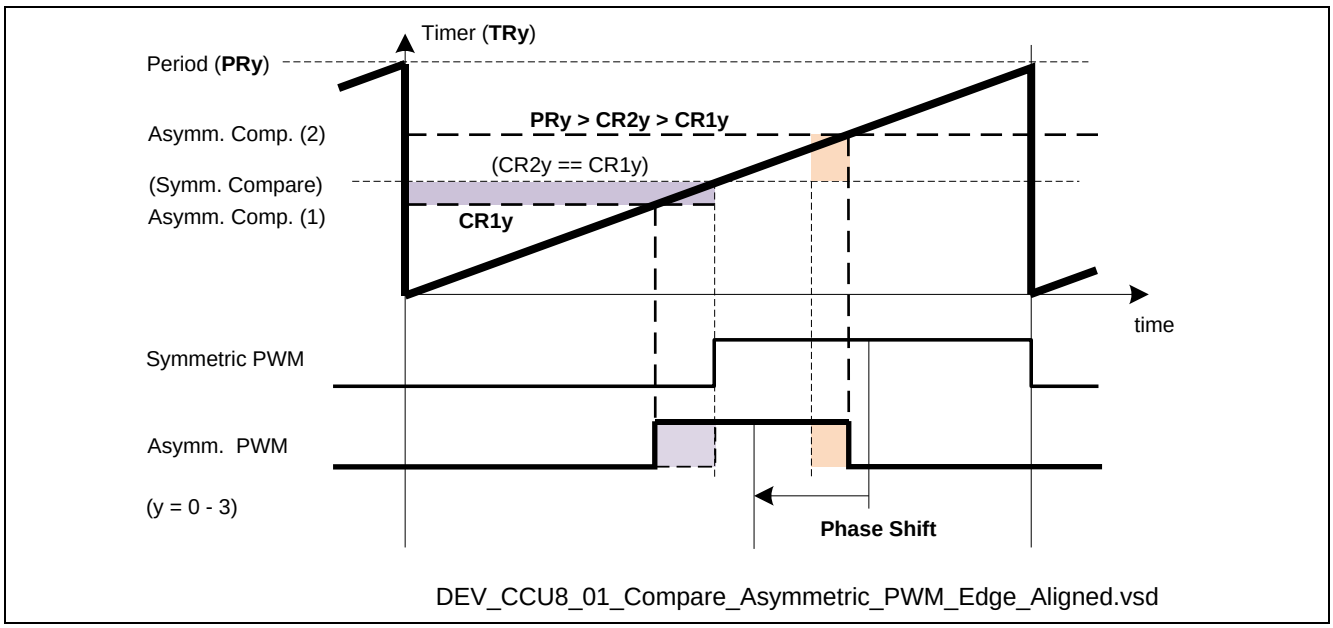


Figure 23 Symmetric PWM and asymmetric PWM (Edge Aligned Mode)

Dead-time generation

Each CAPCOM8 timer slice offers two interdependent 8-bit dead-time counters that can generate independent dead time values for rising and falling transitions in the two compare channels. This can be used to prevent a short circuit in the power stage.

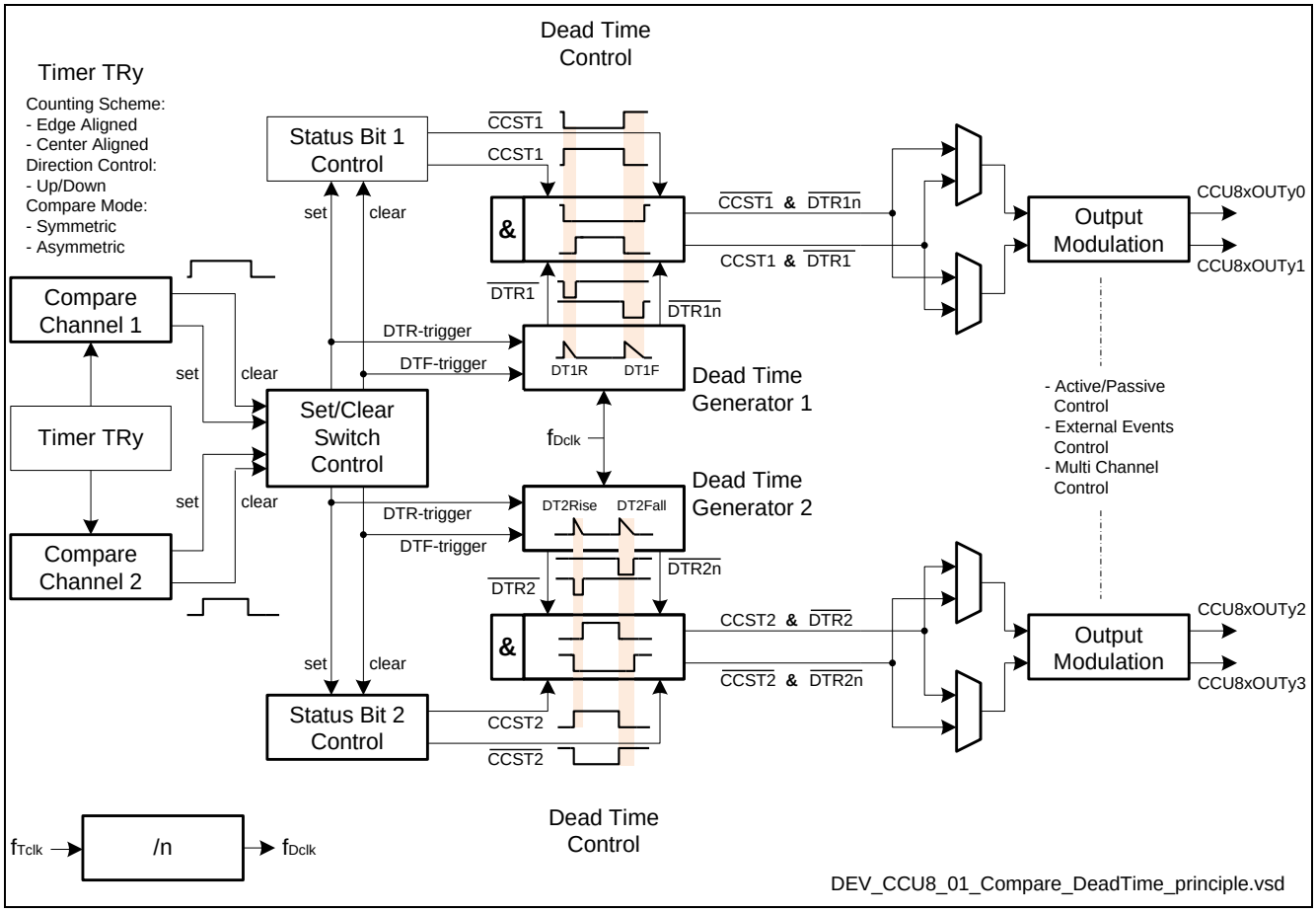


Figure 24 Dead-time generation principle

3.1.1 CCU8 shadow transfer for coherent signal pattern update

All CAPCOM8 timers, in any slice configuration, are assured coherent update by hardware of all relevant timer function parameters. The values in shadow registers are updated on a global preset request simultaneously to all function registers at a period-match or one-match.

3.1.2 The global shadow transfer set enable register

The global register, GCSS, contains all the enable flags that have to be set by software to selectively activate the targeted shadow transfer requests. It can be cleared by hardware after the transfer. The real-time correctness that can be achieved with these logic operations is essential for safe power switching.

3.1.3 Shadow transfer of compare register values

The compare values that are targeted for an update operation have to be written into both the CC8yCR1S/CR2S shadow registers and the corresponding slice transfer set enable bits. For example SySE in GCSS must be preset, at the latest, within the clock cycle of period match (in edge aligned mode) or period/one match (in center aligned mode).

3.1.4 Compound shadow transfers

Besides the compare (CR) values, there is also the timer Period Register (PR) and the PWM output active/passive control bit (PSL) that are updated simultaneously on the SySE flag. The dithering or floating prescaler values can also be simultaneously updated via the SyDSE and SyPSE request flags.

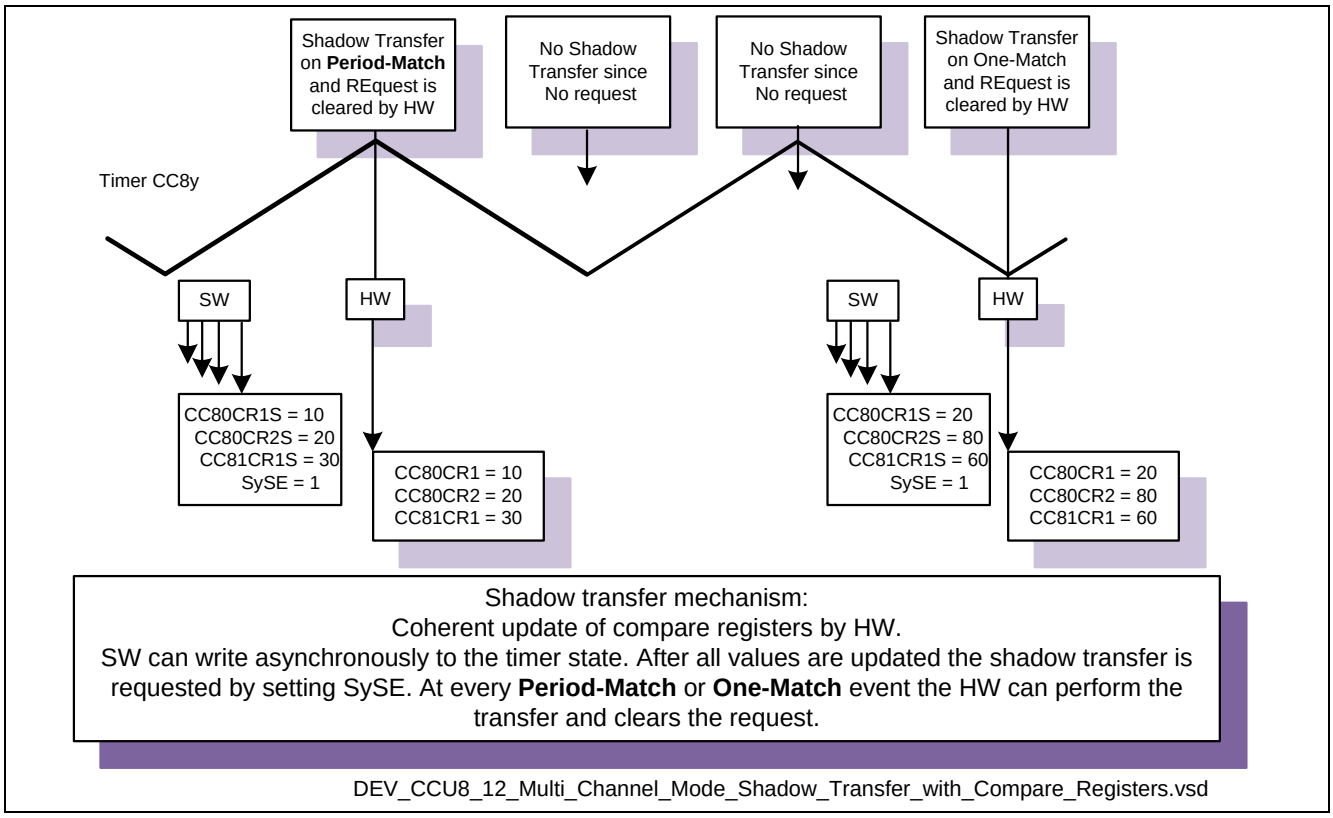
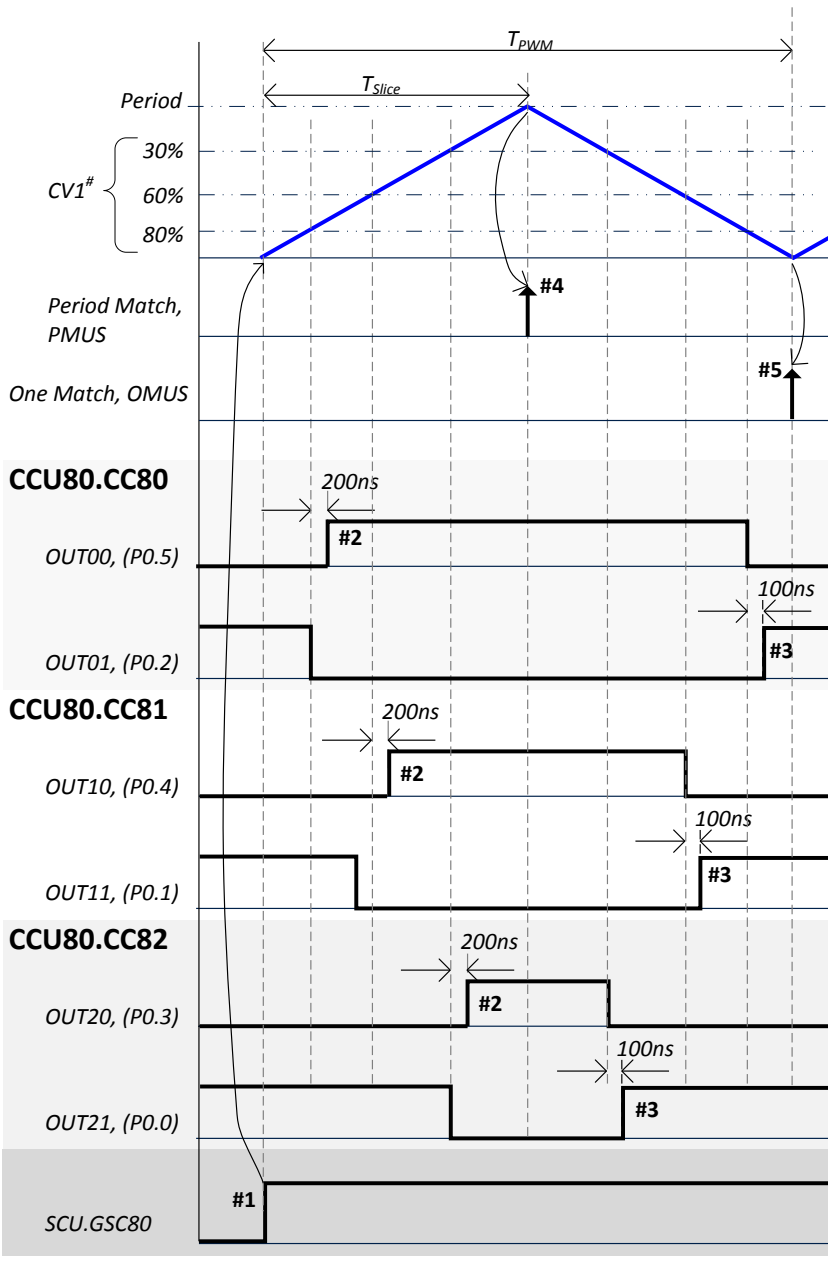


Figure 25 The shadow transfer mechanism (Center Aligned Mode)

3.2 Example application: CCU8 initialization for 3 phase motor drive

Space vector modulation (SVM) is an algorithm for the control of pulse width modulation (PWM) outputs. It is most commonly used to drive 3 phase motor drive. Based upon the angle (sector) and amplitude, it decides which outputs need to be active and the duration (duty cycle) for which they should be in the active state.

In this example, based on the XMC4500, 3 slices of CCU8 are configured to generate 6 PWM outputs that can be used to connect to gate switches. For the purposes of illustration, the frequency of the PWM generated by the 3 slices is set to 20 kHz and compare values are initialized to 30%, 60% and 80% of the slice period value. On Slice 0, 2 interrupt events (period match and one match) are configured and user application code can be added to these routines.



SLICE Configuration:
XMC4500
System Clock = 120 MHz
PWM Frequency = 10 kHz
Mode = Center-aligned

#1: SCU.GSC80 is connected to the input of Event 0 for all 3 Slices of CCU80. This is to allow them to start at the same time. It is set high by writing to the CCUCON SFR. This starts all 3 slices of the CCU80.80, .81, .82 timers on an external start event on Event 0.

#2: Rising edge transition for dead time on Channel 1 is configured to 200ns. This is configured on all 3 slices.

#3: Falling edge transition for dead time on Channel 1 is configured to 100ns.This is configured on all 3 slices.

#4: Period Match Event occurs; An ISR can be triggered to update the duty cycle. Calculation of the new duty cycle value depends on the selected motor control algorithm used in the application.

#5: One Match Event occurs; An ISR can be triggered to restart all the slices if the slices are previously connected in monoshot mode. In this example, this ISR is not required as the slices are all in continuous mode.

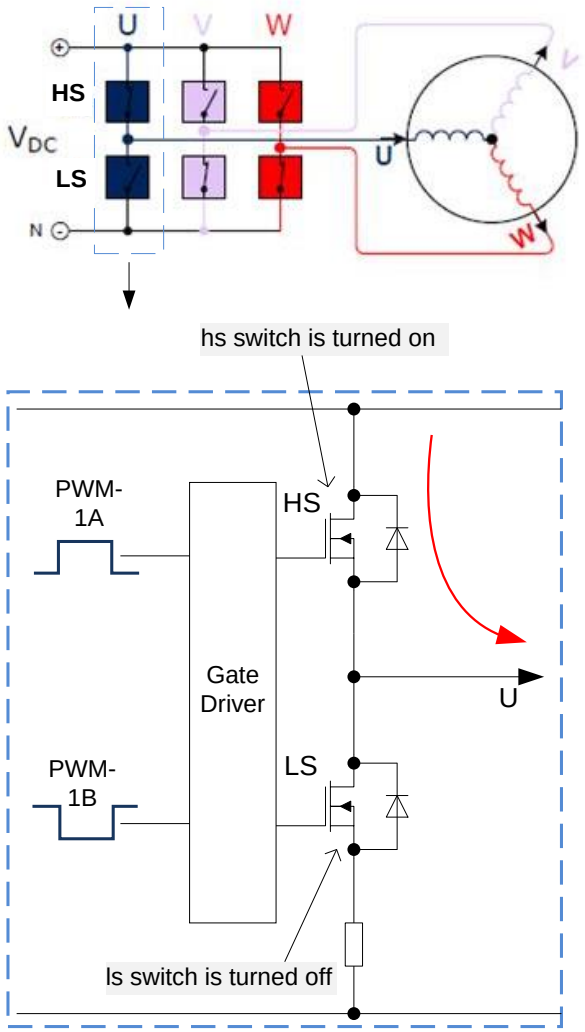
For this example, Compare value for Channel 1 are assigned duty cycle of 30%, 60% and 80%. In an actual application, these are usually assigned based on the selected algorithm.

Figure 26 Example: CCU8 initialization for 3 phase motor drive

3.2.1 Theory of operation

Space Vector Modulation (SVM) is an algorithm for the control of the CCU8 PWM modulation. It is used to control 3 phase motors by modulating the voltage and duty cycle. A three leg voltage source motor contains six MOSFET or IGBTs which act as switches. The switches connected to the positive supply rail are called high side switches (HS) and the switches connected to the negative rail of the power supply are called low side switches (LS). The switches are controlled by PWM inputs. It must be ensured that both switches in the same leg are not turned on at the same time or else the DC supply would be shorted.

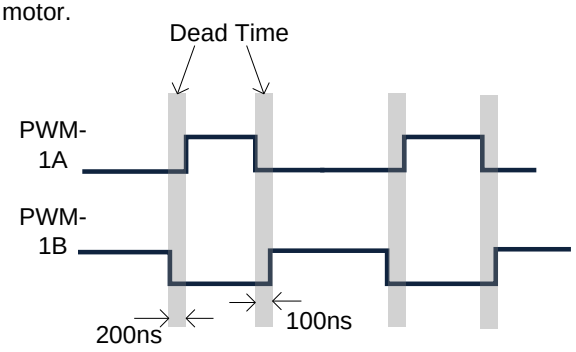
By switching the high side and low side switches on and off, there are eight possible states. These states should not cause cross current but must allow a current following to and from the motor.



Controlling the Switching using PWM
In the figure, a three leg voltage source motor contains six MOSFET or IGBTs which act as switches. The switches are connected to positive supply rail are called high side switches (**HS**) and the switches connected to the negative rail of the power supply are called low side switches (**LS**).

These switches must be controlled to ensure that no time are both switches in the same leg turned on or else the DC supply would be shorted. By switching the high side and low side switches on and off, there are eight possible states. These states should not cause cross current, but allow a current to flow to and from the motor.

Dead-Time Configuration:
Depending on the hardware used, the deadtime must be configured correctly in order to avoid shoot through between high side and low side transistor within the motor.



Example:
Dead Time $_{\text{rising edge}}$ = 200ns
Dead Time $_{\text{falling edge}}$ = 100ns

Figure 27 Controlling the switching using PWM

3.2.2 Deriving the period and compare values

The counting mode has been set to center aligned mode and the clock relationship between f_{PWM}, f_{tclk} and f_{ccu8} is calculated as shown below:

- f_{ccu8} is the frequency of the CCU8 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 this example, the default prescaler factor is 0. This results in a prescaler value of 1 and a timer resolution of 8.33 nS.
- f_{PWM} (the frequency of the PWM signal) is 10 kHz. Given that it is in center aligned mode, the actual slice frequency is two times the PWM signal frequency because it consists of the count up to period match and count down to one match. Hence, the frequency is on the count up to the CCU8_CC80. The PRS register is loaded with the value 5999.

Timer frequency: $f_{tclk} = \frac{f_{ccu8}}{Prescaler}$

Period value: $CCU8_{CC80}.PRS = \frac{f_{tclk}}{2 * f_{PWM}} - 1$

Compare value: $CCU8_{CC80}.CRS = (1 - DC) * (PRS + 1)$

Table 3 Calculated prescaler factor, period and compare values

Type	Calculated value
Prescaler factor	$2^0 = 1$
Period @20 kHz frequency	5999
Compare value @80% Duty Cycle	1200
Compare value @60% Duty Cycle	2400
Compare value @30% Duty Cycle	4200

3.2.3 Deriving the dead-time

The dead time for the rising edge and falling edge is calculated as shown below:

- f_{tclk} is the timer resolution used to increment a timer counter. Each timer slice supports a dedicated prescaler value selector. In this example, a prescaler value of 1 and a timer resolution of 8.33 ns, is used.
- DTCC is the divider factor for the prescaler clock configuration of the dead time counter. It supports divider factor on f_{tclk} of 1/2/4/8.
- f_{dclk} is the frequency of the deadtime clock generator.

Dead time clock: $f_{dclk} = \frac{f_{tclk}}{DTCC}$

Dead time counter: DTxR, DTxF $= Desired\ Dead\ time * f_{dclk}$

Table 4 Calculated dead-time values

Type	Calculated value
DTCC, prescaler divider factor	1
DT1R (Rise value for dead time of 200ns on channel 1)	24
DT1F (Fall value for dead time of 100ns on channel 1)	12

3.2.4 Macro and variable settings

XMC™ Lib project includes:

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

Project macro definitions for CCU8:

```
#define MODULE_PTR              CCU80
#define MODULE_NUMBER           (0U)

#define SLICE0_PTR              CCU80_CC80
#define SLICE0_NUMBER           (0U)
#define SLICE0_OUTPUT00         P0_5
#define SLICE0_OUTPUT01         P0_2

#define SLICE1_PTR              CCU80_CC81
#define SLICE1_NUMBER           (1U)
#define SLICE1_OUTPUT10         P0_4
#define SLICE1_OUTPUT11         P0_1

#define SLICE2_PTR              CCU80_CC82
#define SLICE2_NUMBER           (2U)
#define SLICE2_OUTPUT20         P0_3
#define SLICE2_OUTPUT21         P0_0
```

3.2.5 XMC™ Lib peripheral configuration structure

XMC™ System Clock Unit (SCU) configuration:

```
/* XMC Clock configuration structure */
XMC_SCU_CLOCK_CONFIG_t clock_config =
{
  .syspll_config.n_div = 80U,
  .syspll_config.p_div = 2U,
  .syspll_config.k_div = 4U,
  .syspll_config.mode = XMC_SCU_CLOCK_SYSPLL_MODE_NORMAL,
  .syspll_config.clksrc = XMC_SCU_CLOCK_SYSPLLCLKSRC_OSCHP,
  .enable_oschp = true,
  .enable_osculp = false,
  .calibration_mode = XMC_SCU_CLOCK_FOFI_CALIBRATION_MODE_FACTORY,
  .fstdby_clksrc = XMC_SCU_HIB_STDBYCLKSRC_OSI,
  .fsys_clksrc = XMC_SCU_CLOCK_SYSCLKSRC_PLL,
  .fsys_clkdiv = 1U,
  .fcpu_clkdiv = 1U,
  .fccu_clkdiv = 1U,
  .fperipheral_clkdiv = 1U
};
```

XMC™ Capture/Compare Unit 8 (CCU8) cConfiguration for the 3 Slices:

```
XMC_CCU8_SLICE_COMPARE_CONFIG_t SLICE_config     =
{
```

Multi phase output pattern generation

```
      .timer_mode             = (uint32_t)XMC_CCU8_SLICE_TIMER_COUNT_MODE_CA,
      .monoshot               = (uint32_t)XMC_CCU8_SLICE_TIMER_REPEAT_MODE_REPEAT,
      .shadow_xfer_clear      = 0U,
      .dither_timer_period    = 0U,
      .dither_duty_cycle      = 0U,
      .prescaler_mode         = (uint32_t)XMC_CCU8_SLICE_PRESCALER_MODE_NORMAL,
      .mcm_ch1_enable         = 0U,
      .mcm_ch2_enable         = 0U,
      .slice_status           = (uint32_t)XMC_CCU8_SLICE_STATUS_CHANNEL_1,
      .passive_level_out0     = (uint32_t)XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
      .passive_level_out1     = (uint32_t)XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
      .passive_level_out2     = (uint32_t)XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
      .passive_level_out3     = (uint32_t)XMC_CCU8_SLICE_OUTPUT_PASSIVE_LEVEL_LOW,
      .asymmetric_pwm         = 0U,
      .invert_out0            = 0U,
      .invert_out1            = 1U,
      .invert_out2            = 0U,
      .invert_out3            = 1U,
      .prescaler_initval      = 0U,
      .float_limit            = 0U,
      .dither_limit           = 0U,
      .timer_concatenation    = 0U,
    };

XMC_CCU8_SLICE_EVENT_CONFIG_t SLICE_event0_config =
{
      .mapped_input     = XMC_CCU8_SLICE_INPUT_H,       //Connected to SCU.GSC80
      .edge             = XMC_CCU8_SLICE_EVENT_EDGE_SENSITIVITY_RISING_EDGE,
      .level            = XMC_CCU8_SLICE_EVENT_LEVEL_SENSITIVITY_ACTIVE_LOW,
      .duration         = XMC_CCU8_SLICE_EVENT_FILTER_DISABLED,
};

/* Dead time configuration structure*/
XMC_CCU8_SLICE_DEAD_TIME_CONFIG_t SLICE_dt_config =
{
      .enable_dead_time_channel1        = 1U,
      .enable_dead_time_channel2        = 0U,
      .channel1_st_path                 = 1U,
      .channel1_inv_st_path             = 1U,
      .channel2_st_path                 = 0U,
      .channel2_inv_st_path             = 0U,
      .div                              = (uint32_t) XMC_CCU8_SLICE_DTC_DIV_1,
      .channel1_st_rising_edge_counter  = 24U,  //200ns
      .channel1_st_falling_edge_counter = 12U,  //100ns
      .channel2_st_rising_edge_counter  = 0U,   //0ns
      .channel2_st_falling_edge_counter = 0U,   //0ns
};
```

XMC™ GPIO configuration:

```
// Configuration for A2 class pads: Port0.2, 0.3, 0.4, 0.5
XMC_GPIO_CONFIG_t OUTPUT_strong_sharp_config =
{
        .mode                = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3,
        .output_level        = XMC_GPIO_OUTPUT_LEVEL_LOW,
        .output_strength     = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SHARP_EDGE
};

// Configuration for A1+ class pads: Port0.0, 0.1
XMC_GPIO_CONFIG_t OUTPUT_strong_soft_config =
{
       .mode            = XMC_GPIO_MODE_OUTPUT_PUSH_PULL_ALT3,
       .output_level    = XMC_GPIO_OUTPUT_LEVEL_LOW,
       .output_strength = XMC_GPIO_OUTPUT_STRENGTH_STRONG_SOFT_EDGE
};
```

3.2.6 Interrupt service routine function implementation

The CCU80 interrupt handler function for period match or one match event. In the interrupt routine, the specific application code can be added as needed:

```
/* Period Match and One Match ISR Handler */
void CCU80_0_IRQHandler(void)
{
        if(XMC_CCU8_SLICE_GetEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH))
        {
                XMC_CCU8_SLICE_ClearEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);

                /*      Application specific code */
        }

        if(XMC_CCU8_SLICE_GetEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_ONE_MATCH))
        {
                XMC_CCU8_SLICE_ClearEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_ONE_MATCH);

                /*      Application specific code */
        }
}
```

3.2.7 Main function implementation

Before the start and execution of timer slice software for the first time, the CCU8 must be initialized appropriately using the following sequence:

- Clock setup:

```
/* Ensure clock frequency is set at 120 MHz */
XMC_SCU_CLOCK_Init(&clock_config);
```

- Enable clock, enable prescaler block and configure global control:

```
/* Enable CCU8 module */
```

```
XMC_CCU8_Init(MODULE_PTR, XMC_CCU8_SLICE_MCMS_ACTION_TRANSFER_PR_CR);

/* Start the prescaler */
XMC_CCU8_StartPrescaler(MODULE_PTR);

/* Ensure fCCU reaches CCU80 */
XMC_CCU8_SetModuleClock(MODULE_PTR, XMC_CCU8_CLOCK_SCU);
```

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

```
/* Configure CCU8x_CC8y slice as timer */
XMC_CCU8_SLICE_CompareInit(SLICE0_PTR, &SLICE_config);
XMC_CCU8_SLICE_CompareInit(SLICE1_PTR, &SLICE_config);
XMC_CCU8_SLICE_CompareInit(SLICE2_PTR, &SLICE_config);

/* Set period match value of the timer  */
XMC_CCU8_SLICE_SetTimerPeriodMatch(SLICE0_PTR, 5999U);
XMC_CCU8_SLICE_SetTimerPeriodMatch(SLICE1_PTR, 5999U);
XMC_CCU8_SLICE_SetTimerPeriodMatch(SLICE2_PTR, 5999U);

/* Set timer compare match value for channel 1 - (80%, 60%, 30%) Duty Cycle */
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE0_PTR, \
        XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 1200U);
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE1_PTR, \
        XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 2400U);
XMC_CCU8_SLICE_SetTimerCompareMatch(SLICE2_PTR, \
        XMC_CCU8_SLICE_COMPARE_CHANNEL_1, 4200U);

/* Transfer value from shadow timer registers to actual timer registers */
XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_0);
XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_1);
XMC_CCU8_EnableShadowTransfer(MODULE_PTR, XMC_CCU8_SHADOW_TRANSFER_SLICE_2);

/* Configure events */
XMC_CCU8_SLICE_ConfigureEvent(SLICE0_PTR, XMC_CCU8_SLICE_EVENT_0, \
        &SLICE_event0_config);
XMC_CCU8_SLICE_ConfigureEvent(SLICE1_PTR, XMC_CCU8_SLICE_EVENT_0, \
        &SLICE_event0_config);
XMC_CCU8_SLICE_ConfigureEvent(SLICE2_PTR, XMC_CCU8_SLICE_EVENT_0, \
        &SLICE_event0_config);

XMC_CCU8_SLICE_StartConfig(SLICE0_PTR, XMC_CCU8_SLICE_EVENT_0, \
        XMC_CCU8_SLICE_START_MODE_TIMER_START_CLEAR);
XMC_CCU8_SLICE_StartConfig(SLICE1_PTR, XMC_CCU8_SLICE_EVENT_0, \
        XMC_CCU8_SLICE_START_MODE_TIMER_START_CLEAR);
XMC_CCU8_SLICE_StartConfig(SLICE2_PTR, XMC_CCU8_SLICE_EVENT_0, \
        XMC_CCU8_SLICE_START_MODE_TIMER_START_CLEAR);

/* Enable events */
XMC_CCU8_SLICE_EnableEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH);
XMC_CCU8_SLICE_EnableEvent(SLICE0_PTR, XMC_CCU8_SLICE_IRQ_ID_ONE_MATCH);
```

```
/* Connect period match and one match event to SR0 */
XMC_CCU8_SLICE_SetInterruptNode(SLICE0_PTR, \
        XMC_CCU8_SLICE_IRQ_ID_PERIOD_MATCH, XMC_CCU8_SLICE_SR_ID_0);

/* Set NVIC priority */
NVIC_SetPriority(CCU80_0_IRQn, 63U);

/* Enable IRQ */
NVIC_EnableIRQ(CCU80_0_IRQn);

/* Deadtime initialisation*/
XMC_CCU8_SLICE_DeadTimeInit(SLICE0_PTR, &SLICE_dt_config);
XMC_CCU8_SLICE_DeadTimeInit(SLICE1_PTR, &SLICE_dt_config);
XMC_CCU8_SLICE_DeadTimeInit(SLICE2_PTR, &SLICE_dt_config);

/*Initializes the GPIO*/
XMC_GPIO_Init(SLICE0_OUTPUT00, &OUTPUT_strong_sharp_config);
XMC_GPIO_Init(SLICE0_OUTPUT01, &OUTPUT_strong_sharp_config);

XMC_GPIO_Init(SLICE1_OUTPUT10, &OUTPUT_strong_sharp_config);
XMC_GPIO_Init(SLICE1_OUTPUT11, &OUTPUT_strong_soft_config);

XMC_GPIO_Init(SLICE2_OUTPUT20, &OUTPUT_strong_sharp_config);
XMC_GPIO_Init(SLICE2_OUTPUT21, &OUTPUT_strong_soft_config);
```

- Start timer running on external start event:

```
/* Get the slice out of idle mode */
XMC_CCU8_EnableClock(MODULE_PTR, SLICE0_NUMBER);
XMC_CCU8_EnableClock(MODULE_PTR, SLICE1_NUMBER);
XMC_CCU8_EnableClock(MODULE_PTR, SLICE2_NUMBER);

/* Start the PWM on a rising edge on SCU.GSC80 */
XMC_SCU_SetCcuTriggerHigh(XMC_SCU_CCU_TRIGGER_CCU80);
```

4 Revision history

Current version is Revision 1.1, 2016-02

Page or reference	Description of change
V1.0, 2015-07	
	Initial version
V1.1, 2016-02	
	Updated Section 3.2.5 to set up the period match and one match interrupt handler and events

www.infineon.com

Edition 2016-02

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

AP32288