How to use analog comparators in PSOC™ Control C3 MCU

About this document

Scope and purpose

This application note introduces the features of the low-power comparator and slope generator (CSG), and then describes how to use these analog comparators in the PSOC™ Control C3 family MCU.

Intended audience

This document is intended for anyone who uses the analog comparators of the PSOC™ Control C3 MCUs.

This application note assumes that you are familiar with PSOC™ Control C3 and the ModusToolbox™ development environment.

If you are new to PSOC™ Control C3, see *AN238329 - Getting started with PSOC™ Control C3 MCU on ModusToolbox™*. If you are new to ModusToolbox™, see the ModusToolbox™ home page.

Associated part family

All PSOC™ Control C3 devices.

Table of contents

1 Introduction

The PSOC™ Control C3 MCU provides two low-power comparators and five active analog comparators. The low-power comparators can operate in all power modes, which allow other analog system resources to be disabled while retaining the ability to monitor external voltage levels during Deep Sleep and Hibernate modes. The active analog comparators are part of the high-performance programmable analog subsystem (HPPASS) which operates in Active mode only.

Figure 1 shows the relationship between the analog input levels and the digital output. When the comparator analog positive input is less than the analog negative input, the output of the comparator is at the digital **L** level. When the comparator analog positive input is greater than the analog negative input, the output of the comparator is at the digital **H** level.

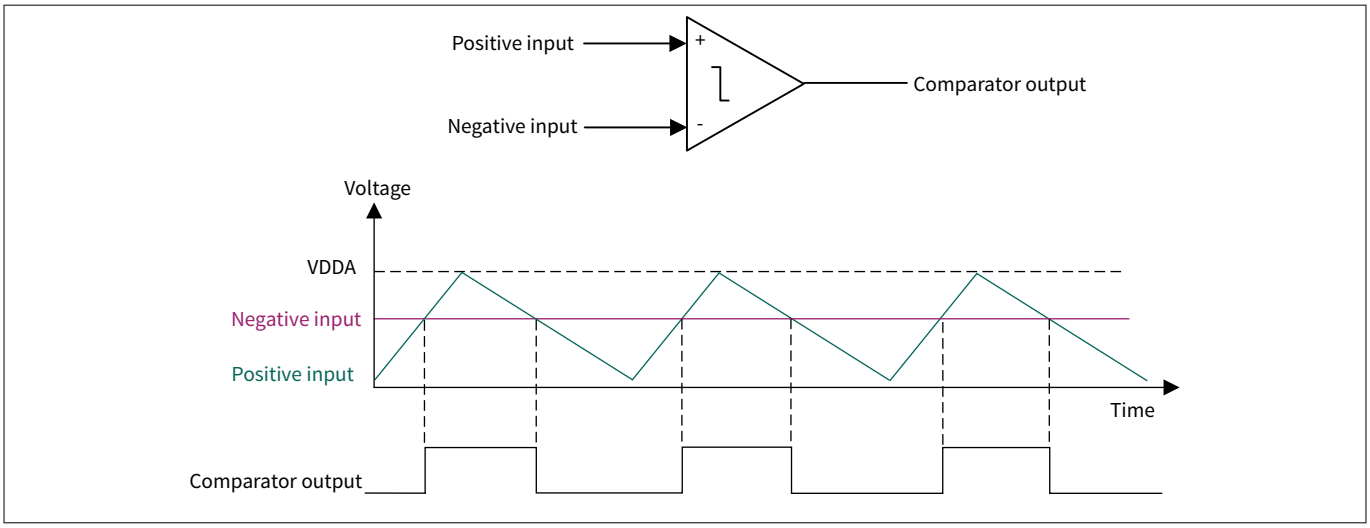

Figure 1 Comparator overview

This application note introduces the features of these analog comparators, and describes how to use them.

2 Overview of analog comparators

This section describes the features of the low-power comparator and active comparator (comparator and slope generator) in PSOC™ Control C3 MCU.

2.1 Low-power comparators (LPCOMP)

PSOC™ Control C3 MCU has two low-power comparators. The main purpose of these comparators is to offer fast detection of a voltage change in normal operating modes and ultra-low power operation in all system power modes.

The low-power comparator output can be:

- Inspected by the CPU
- Used as an interrupt/wake-up source to the CPU when in CPU Sleep mode
- Used as a wake-up source to system resources when in system Deep Sleep mode or Hibernate mode
- Fed to a GPIO or trigger multiplexer as an asynchronous or synchronous signal (level or pulse)

2.1.1 Overview

The following figure shows the block diagram for the low-power comparator.

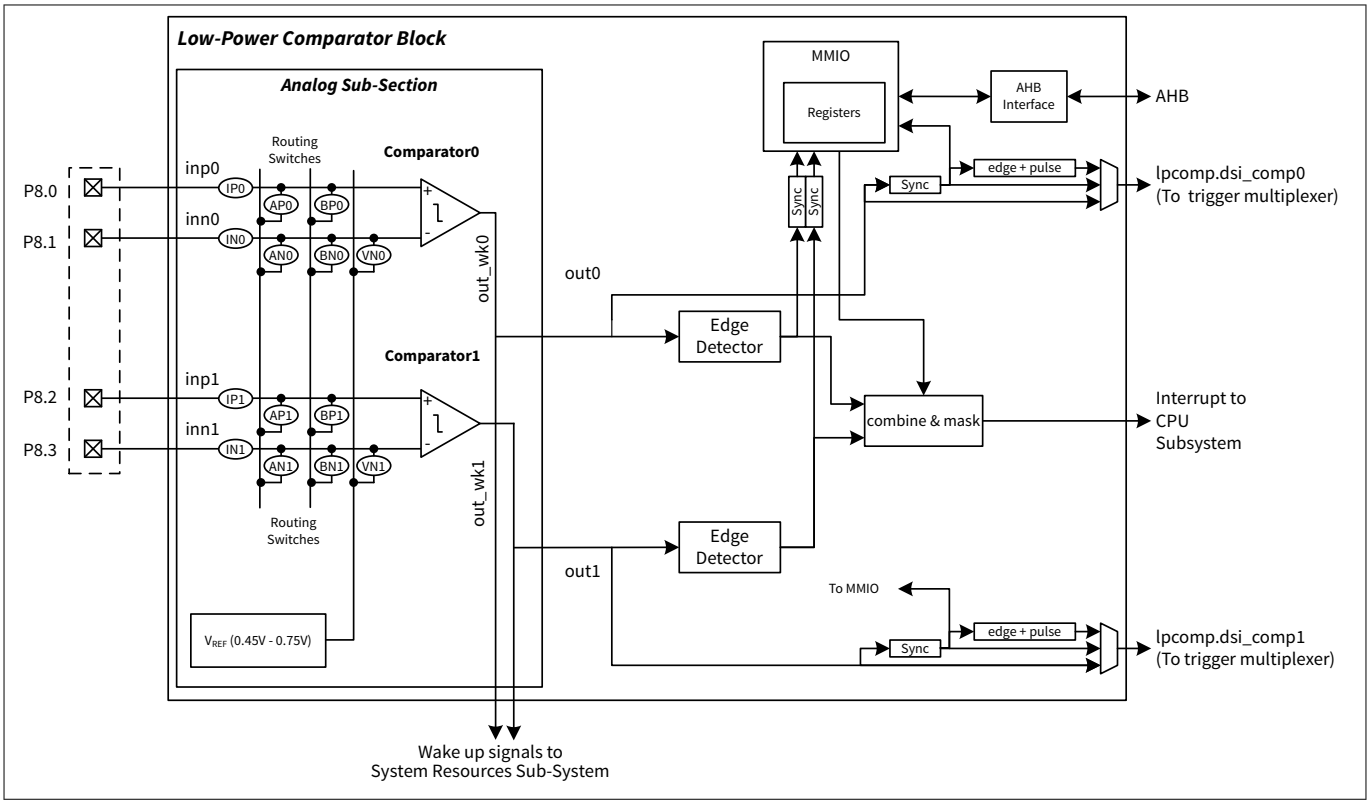

Figure 2 Low-power comparator block diagram

On a higher level, the block consists of two subsections:

- Analog logic
- Digital logic

The analog subsection consists of two comparators with the option of hysteresis switches to connect to inputs and the reference voltage source. The output from the comparators is driven to the system resources subsystem (SRSS) directly for device wake-up from Hibernate mode.

The comparator output is also driven to the digital subsection. This subsection provides different output options, such as synchronous level and pulse output, and options to trigger interrupts on positive, negative, or both edges of the comparator outputs to wake-up the device from Sleep and Deep Sleep modes.

2.1.2 Input and output configuration

Low-power comparators operate with the following input options:

- Compare two voltages from external pins
- Compare external signals with a locally generated reference voltage. Note that this voltage is not a precision reference and can vary from 0.4 V to 0.8 V. Make sure that external input voltage sensing is less than 0.4 V or greater than 0.8 V

The internal routing switches (IP0, AP0, BP0, IN0, AN0, BN0, VN0, IP1, AP1, BP1, IN1, AN1, BN1, VN1) shown in the block diagram are controlled using the LPCOMP_CMP0_SW, LPCOMP_CMP1_SW, LPCOMP_CMP0_SW_CLEAR, and LPCOMP_CMP1_SW_CLEAR registers.

The output can be configured to one of the following options using the control registers (LPCOMP_CMP0_CTRL and LPCOMP_CMP1_CTRL):

- Direct comparator output
- Synchronous comparator output-level
- Synchronous comparator output-pulse

The DSI_BYPASS0 and DSI_BYPASS1 bits of the control register select the direct or synchronous comparator output. The DSI_LEVEL0 and DSI_LEVEL1 bits of the control register select between pulse or level output. Pulse output is two clk_sys cycles, and it can be generated on a positive edge, a negative edge, or on both edges configured using the INTTYPE0 and INTTYPE1 bits of the control register.

In addition, comparator outputs can be routed to GPIOs and other on-chip peripherals such as DMA and TCPWMs using the trigger multiplexer.

For more information about LPCOMP input and output configuration, see the *Input configuration* and *Output configuration* sections of the PSOC™ Control C3 architecture reference manual.

2.1.3 Power mode and speed configuration

The low comparator is the only analog block capable of operating even in Hibernate mode. It provides an option to generate an interrupt to wake up the device. The block provides three programmable power levels: normal, low, and ultra-low.

The power or speed setting for comparator 0/1 is configured using the MODE0/1 bitfield of the LPCOMP_CMPx_CTRL register, where x = 0, 1, respectively.

The power consumption and response time vary depending on the selected power mode; power consumption is the highest in fast mode and the lowest in ultra-low-power mode. Similarly, the response time is the fastest in fast mode and the slowest in ultra-low-power mode. The response time of the block increases with the decrease in the power level as shown in Table 1.

Table 1 Low-power comparator modes

Power mode	Current consumption – maximum (µA)	Response time – maximum (µs)
Normal	150	0.1
Low-power	10	1
Ultra-low-power	0.85	20

2.1.4 Hysteresis

For applications that compare signals close to each other and slow-changing signals, hysteresis helps to avoid oscillations at the comparator output when the signals are noisy. For such applications, a fixed hysteresis may be enabled in the comparator block. See the device datasheet for the hysteresis voltage range.

For comparator 0 and comparator 1, the hysteresis level is enabled/disabled using the HYST0 and HYST1 bitfields in the LPCOMP_CMP0_CTRL and LPCOMP_CMP1_CTRL registers. Figure 3 shows the waveform of comparator output with hysteresis enabled.

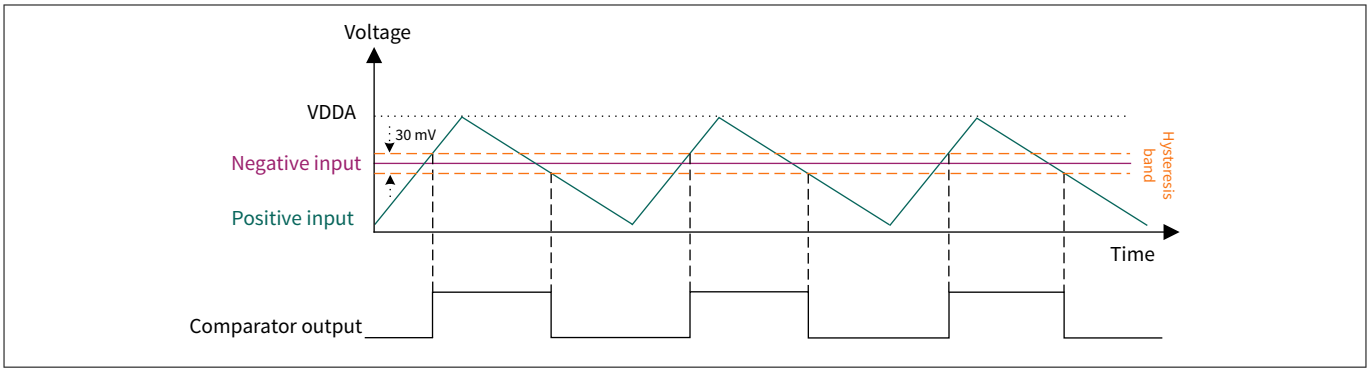

Figure 3 Waveform of comparator output with hysteresis

2.1.5 Wake up from low-power modes

The comparator is operational in the device's low-power modes, including Sleep, DeepSleep, and Hibernate modes. The comparator output can wake the device from Sleep and DeepSleep modes. The comparator output can generate interrupts in DeepSleep mode when enabled in the LPCOMP_CONFIG register. Do not set the INTTYPEx bits in the LPCOMP_CMPx_CTRL register to disabled, and set the INTR_MASKx bit in the LPCOMP_INTR_MASK register for the corresponding comparator to wake the device from low-power modes.

The comparators have the following restrictions when operating in DeepSleep and Hibernate modes:

- Comparators should be operated in ultra-low-power mode
- Input to the comparator should come from the dedicated pins. The internal reference, VREF, is available and can be connected to the inverting inputs of the comparator

***Note**: Set the LPREF_EN bit in the LPCOMP_CONFIG register to '1', to use the block in DeepSleep and Hibernate modes. This bit controls the local reference voltage VREF as well as the bias current required for the block's operation in low-power modes.*

2.2 Comparator and slope generator (CSG)

PSOC™ Control C3 MCU contains a high-performance programmable analog subsystem (HPPASS). It has five comparator and slope generators (CSG). The CSG is a flexible block, which compares a 10-bit, 30 Msps DAC value with a selected analog input signal or compares two analog inputs. The output of this block is the 1-bit digital compare value that can be routed to a GPIO or TCPWM (through TriggerMux). For example, as an input to terminate the PWM signal if an overvoltage or overcurrent condition is detected.

2.2.1 Overview

Figure 4 shows the block diagram for the CSG. There are two logic partitions wrapped around the CSG analog subblock: one for the DAC code generation, and the other for comparator output post processing.

Input and output trigger also play a key role in the CSG function, as well as the user interaction via the AHB bus. The following diagram is considered a **slice**. The PSOC™ Control C3 MCU supports five slices.

2 Overview of analog comparators

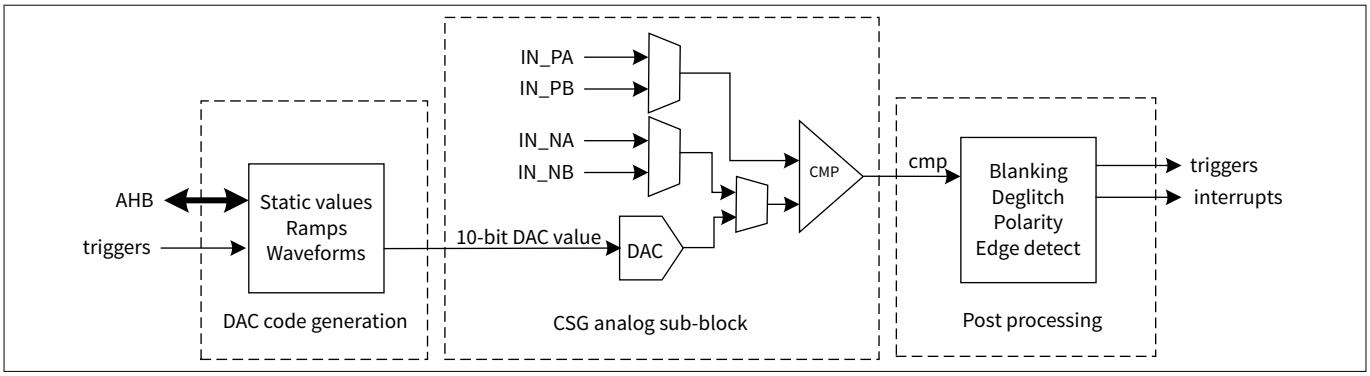

Figure 4 CSG block diagram

Every CSG slice comparator has different positive and negative inputs by setting the CMP_POS_SEL and CMP_NEG_SEL fields of the HPPASS_CSG_SLICEx_CMP_CFG (x = 0 - 4) register (Table 2 lists the comparators of all inputs for every slice).

Table 2 CSG comparator positive and negative inputs

Slice	Comparator positive inputs	Comparator negative inputs
Slice 0	AN_A0/AN_B0	DAC0/AN_A1/AN_B1
Slice 1	AN_A1/AN_B1	DAC1/AN_A2/AN_B2
Slice 2	AN_A2/AN_B2	DAC2/AN_A3/AN_B3
Slice 3	AN_A3/AN_B3	DAC3/AN_A4/AN_B4
Slice 4	AN_A4/AN_B4	DAC4

2.2.2 10-bit DAC

Each CSG slice consists of a 10-bit, 30 Msps DAC to generate the comparator reference. There are various modes of generating and controlling the DAC codes, both static and dynamic. It contains the following modes:

- Direct firmware control of DAC code
- Buffered firmware control of DAC code
- Two-value (Hysteretic) control of DAC code
- Automatic DAC ramp generation
- Optional use of lookup table (LUT) for DAC arbitrary waveform generation

For details of DAC modes, see the *CSG DAC modes* section of the PSOC™ Control C3 architecture reference manual.

2.2.2.1 DAC mode start/stop

A mode may only be changed when there is no DAC operation currently in progress.

There are two ways to start a new DAC mode firmware or hardware (see the Registers Technical Reference Manual for the details of the HPPASS_CSG_SLICEx_DAC_MODE_START register):

- **HW_START**: When this bit is set, the DAC mode start is synchronized to the next DAC_TR_START selected trigger, set DAC_TR_START_SEL fields of HPPASS_CSG_SLICEx_DAC_CFG register to select the DAC start trigger, which can be any of the external hardware trigger inputs or AC trigger.

  ***Note**: if the hardware start trigger is asserted again while the DAC mode is in progress, it will be ignored*

- **FW_START**: The mode is enabled immediately when this bit is set

The HW_START/FW_START bit of HPPASS_CSG_SLICEx_DAC_MODE_START register is automatically reset to '0' on DAC_SLOPE_DONE event, unless HPPASS_CSG_SLICEx_DAC_CFG.DAC_CONT bit is set for continuous mode.

To stop the current mode's progress, clear the associated start bit. The current DAC_VAL is frozen to the last value, until it is overwritten directly, or a new mode is started.

In addition to disabling the mode by resetting the HW_START and FW_START bit to 0, any direct write to the HPPASS_CSG_SLICEx_DAC_VAL register overrides the current DAC output and disables any DAC mode in progress.

The HPPASS_CSG_SLICEx_DAC_MODE_START register has a read-only BUSY bit to indicate that a DAC mode has been enabled and is currently in progress. If FW_START is used to start the mode, then the FW_START bit is equivalent to this bit and low when the mode is completed or stopped. If HW_START is used to start the mode, then this status bit goes high on the DAC_START_TR event, and low when the mode is completed or stopped.

2.2.2.2 DAC update

Set the DAC_TR_UPDATE_SEL bitfields of the HPPASS_CSG_SLICEx_DAC_CFG register to select the DAC update trigger. When the selected DAC_TR_UPDATE is asserted, the DAC code is updated. The behavior depends on the selected DAC_MODE.

The following triggers are available for the DAC update trigger:

- Any of the eight HPPASS hardware triggers from other on-chip peripherals or firmware trigger
- CSG slice local DAC_PERIOD counter (the period divider is programmed in the HPPASS_CSG_SLICEx_DAC_PERIOD register)
- CSG slice local comparator output or inverted local comparator output (used only in DAC Hysteresis mode)
- AC trigger

2.2.3 Post-processing

Figure 5 shows the diagram of the hardware resources in CSG comparator post-processing.

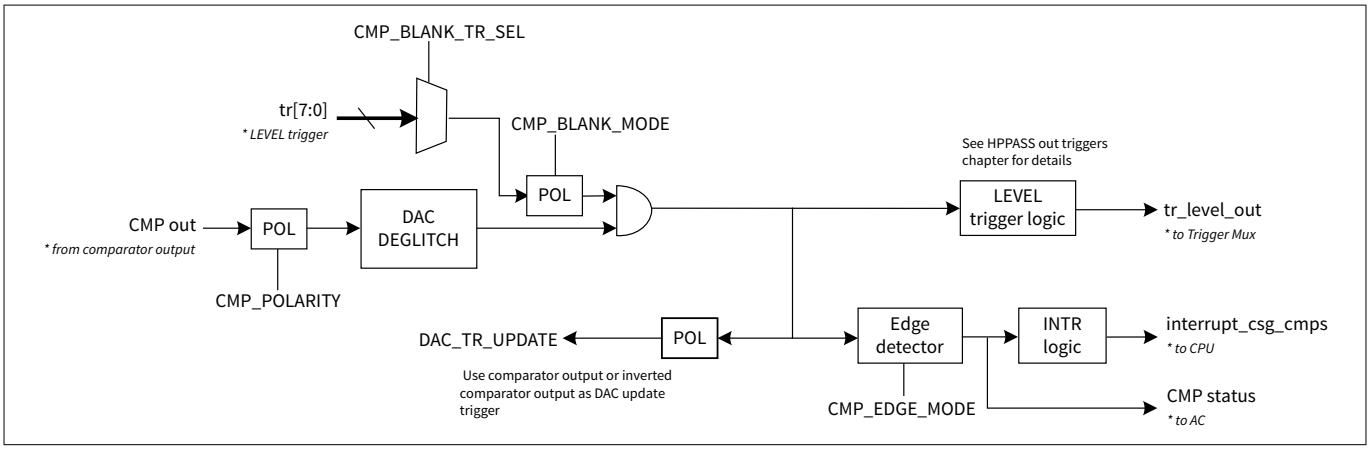

Figure 5 CSG comparator post-processing

The main features of CSG comparator post-processing are as follows:

- Configuring of the comparator output polarity
- Blanking the comparator, gating it to ‘0’ with an input trigger
- Deglitching the comparator output that may occur due to a DAC code change; the comparator output is latched for a programmable number of clock cycles

- Generating a configurable (positive edge, negative edge, or both edges) comparator interrupt and AC status
- Ability to select the comparator output or inverted comparator output as an update trigger for DAC Hysteresis mode

CMP_POLARITY and CMP_EDGE_MODE bitfields of the HPPASS_CSG_SLICEx_CMP_CFG register configure the comparator output polarity and Edge mode, separately. The CMP_VAL bits of the HPPASS_CSG_SLICEx_CMP_STATUS register read the comparator output value of every CSG slice.

2.2.3.1 Comparator blanking

When the power switches controlled by the PWM are initially turned on, there can be a large current transient that causes the comparator to trip prematurely. Therefore, a comparator blanking capability is provided via an input trigger, which must be a level trigger.

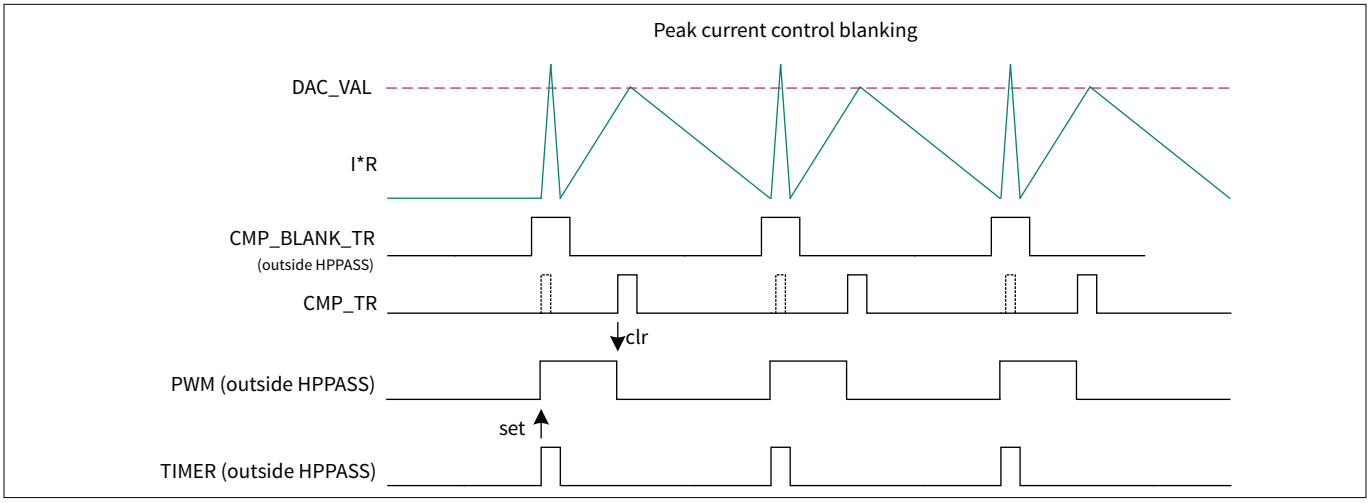

Figure 6 Comparator blanking example

3 Comparator configuration and examples

This section provides examples that help you learn how to use the low-power comparator and HPPASS CSG application. The following examples are based on PSOC™ Control C3 Evaluation Kit hardware. See AN239385 - PSOC™ Control C3 Evaluation Kit guide.

3.1 Comparator with internal DAC reference

The following example demonstrates how to use the HPPASS CSG to compare the external input signal with an internal 10-bit DAC reference in PSOC™ Control C3 MCU devices.

The Figure 7 shows the block diagram. The example uses CSG Slice 0 of the HPPASS to compare the external input voltage from the analog input pin AN_A0 with the internal DAC reference.

Set the DAC operation mode to buffered mode. Alternatively, set the DAC output buffer to low threshold 0.8 V and high threshold 2 V when the DAC buffer is empty. Configure the TCPWM overflow trigger to trigger DAC mode starting and DAC value updating. The CSG comparator output connects to GPIO (LED) by TriggerMux to show comparator level status.

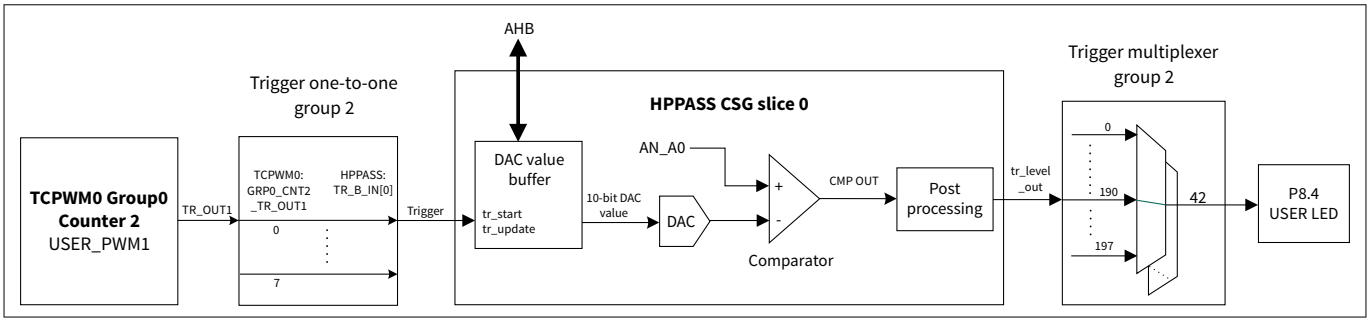

Figure 7 Comparator with internal DAC reference example

ModusToolbox™ is used to configure the HPPASS CSG slice, trigger connections, and TCPWM. Figure 8 shows the configuration of the HPPASS in the Device Configurator in the ModusToolbox™. This involves the following:

- Enables CSG Slice 0
- Configures AN_A0 for comparator positive input
- Configures the internal 10-bit DAC output for comparator negative input
- Sets the DAC operation mode to buffered mode
- Configures HPPASS input trigger 0 to trigger DAC mode start and DAC value buffer update
- Sets HPPASS input trigger 0 to TCPWM trigger out 1
- Sets the HPPASS output level trigger 0 to output comparator level status to LED GPIO

3 Comparator configuration and examples

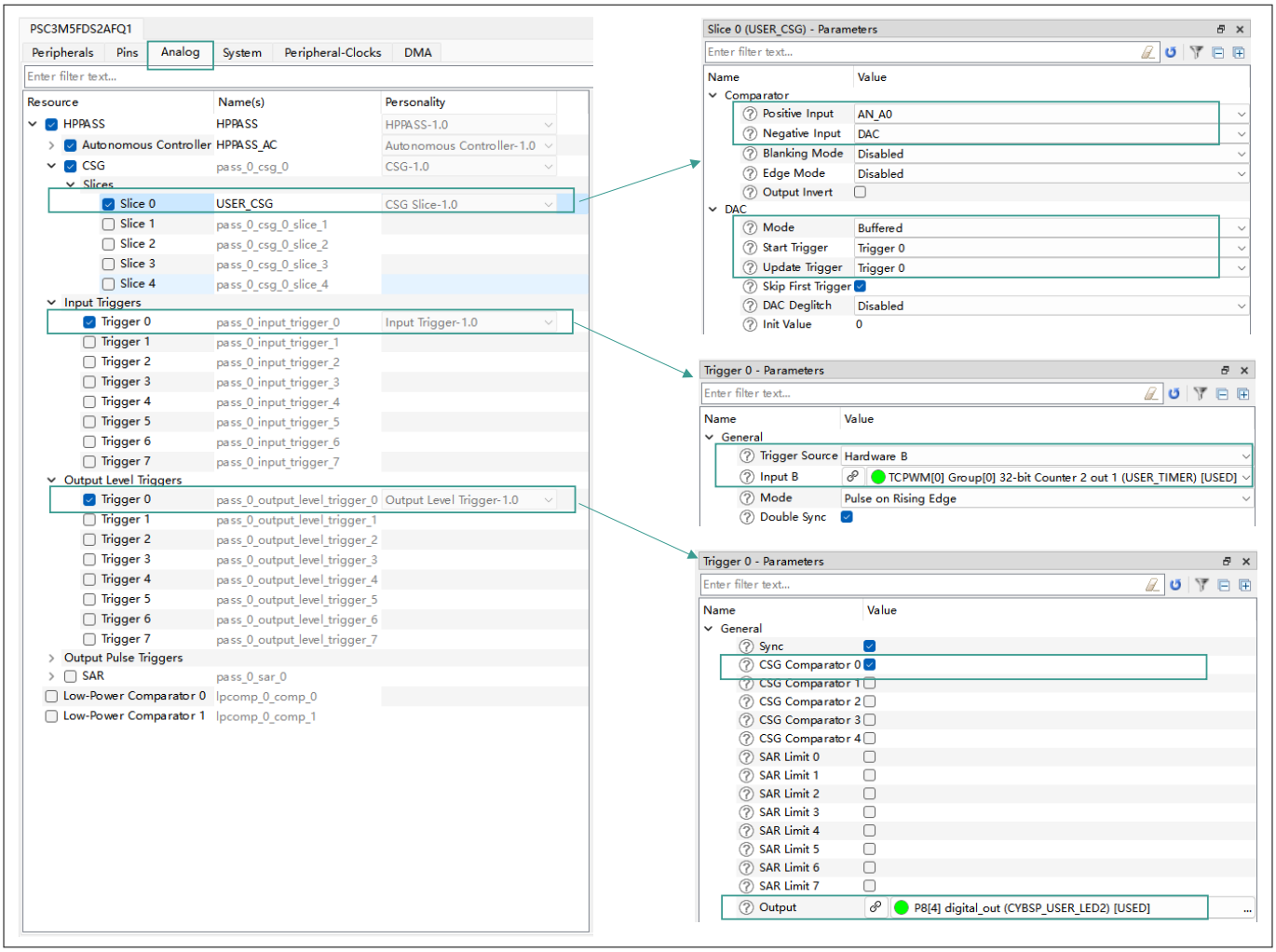

Figure 8 CSG and triggers configuration in ModusToolbox™

Figure 9 shows the TCPWM configuration. This involves the following:

- Configures TCPWM0 Group 0 Counter 2 to counter mode
- Set the period as 1 second
- Enables overflow event for TCPWM trigger output 1
- Connects TCPWM trigger output 1 to HPPASS input trigger 0

3 Comparator configuration and examples

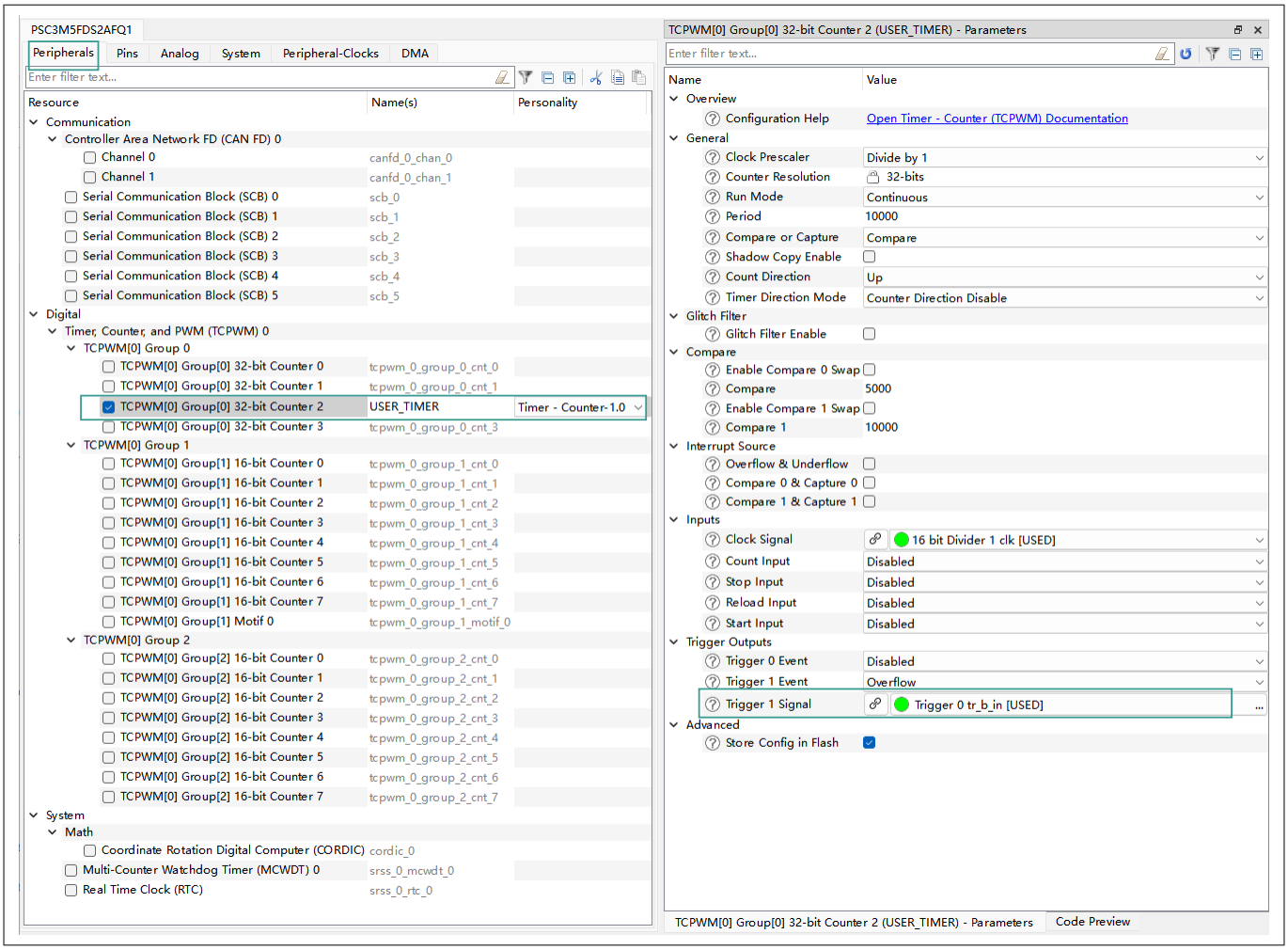

Figure 9 TCPWM configuration in ModusToolbox™

The following code snippet demonstrates the example program. When CSG DAC hardware mode is started, the program alternately sets the DAC output buffer to 0.8 V and 2 V every second after the DAC value buffer is empty.

3 Comparator configuration and examples

Source code of comparator with DAC reference example

```
#include "cyhal.h"
#include "cybsp.h"
#include "cy_pdl.h"
#include "cy_retarget_io.h"

/* The definition of HPPASS AC startup timeout in microseconds.
 * HPPASS startup time is about 5 ms, it contains AREF startup 40us, CSG startup
 * about 15us and SAR ADC calibration time. To be on the safe side, add to 10ms
 */
#define HPPASS_AC_STARTUP_TIMEOUT           (10000U)

/* The low threshold of CSG DAC output: 800mV, value = (1024 * 800mV)/3300mV
 * 10-bit DAC, DAC reference voltage is VDDA and VDDA is 3300mV on EVK board
 */
#define USER_CSG_DAC_OUT_LOW_THRESHOLD      (248u)

/* The high threshold of CSG DAC output: 2000mV, value = (1024 * 2000mV)/3300mV */
#define USER_CSG_DAC_OUT_HIGH_THRESHOLD     (621u)

/* User CSG DAC HW start interrupt mask */
#define USER_CSG_DAC_HW_START_INT_MASK      CY_HPPASS_INTR_CSG_0_DAC_HW_START

/* User CSG DAC buffer empty interrupt mask */
#define USER_CSG_DAC_BUF_EMPTY_INT_MASK     CY_HPPASS_INTR_CSG_0_DAC_BUF_EMPTY

/* The user CSG DAC interrupt configuration structure */
cy_stc_sysint_t user_csg_dac_intr_config =
{
    .intrSrc = pass_interrupt_csg_dac_0_IRQn,
    .intrPriority = 0U,
};

/* The started flag of user CSG DAC */
volatile bool user_csg_dac_is_started = false;

/* The buffer empty flag of user CSG DAC */
volatile bool user_csg_dac_buf_is_empty = false;

/* The DAT output value of User CSG */
volatile uint16_t user_csg_dac_value = 0;

/* Variable for storing character read from terminal */
uint8_t uart_read_value = 0;

/* This function is the user CSG DAC interrupt handler. */
void user_csg_dac_intr_handler(void)
{
    uint32_t intrStatus = Cy_HPPASS_DAC_GetInterruptStatusMasked();
    /* Clear interrupt */
    Cy_HPPASS_DAC_ClearInterrupt(intrStatus);
```

3 Comparator configuration and examples

```
    /* Check CSG DAC HW start interrupt */
    if(USER_CSG_DAC_HW_START_INT_MASK == (intrStatus & USER_CSG_DAC_HW_START_INT_MASK))
    {
        user_csg_dac_is_started = true;
    }
    /* Check CSG DAC buffer empty interrupt */
    if(USER_CSG_DAC_BUF_EMPTY_INT_MASK == (intrStatus & USER_CSG_DAC_BUF_EMPTY_INT_MASK))
    {
        user_csg_dac_buf_is_empty = true;
    }
}

/* This is the main function for CPU */
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init_fc(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
            CYBSP_DEBUG_UART_CTS,CYBSP_DEBUG_UART_RTS,CY_RETARGET_IO_BAUDRATE);
    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize TCPWM using the config structure generated using device configurator*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_Counter_Init(USER_TIMER_HW, USER_TIMER_NUM,
&USER_TIMER_config))
    {
        CY_ASSERT(0);
    }
    /* Enable the initialized TCPWM */
    Cy_TCPWM_Counter_Enable(USER_TIMER_HW, USER_TIMER_NUM);

    /* Configure HPPASS interrupt */
    Cy_HPPASS_DAC_SetInterruptMask(USER_CSG_DAC_HW_START_INT_MASK |
USER_CSG_DAC_BUF_EMPTY_INT_MASK);
    Cy_SysInt_Init(&user_csg_dac_intr_config, user_csg_dac_intr_handler);
    NVIC_EnableIRQ(user_csg_dac_intr_config.intrSrc);

    /* Start the HPPASS autonomous controller (AC) from state 0 */
    if(CY_HPPASS_SUCCESS != Cy_HPPASS_AC_Start(0U, HPPASS_AC_STARTUP_TIMEOUT))
    {
        CY_ASSERT(0);
```

```
    }

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
    printf("PSoC Contorl C3 MCU: HPPASS CSG example\r\n");
    printf("Press 'Enter' key in terminal to start or stop CSG DAC hardware (HW) mode\r\n");

    /* Enable global interrupts */
    __enable_irq();

    /* Start user timer */
    Cy_TCPWM_TriggerStart_Single(USER_TIMER_HW, USER_TIMER_NUM);

    for (;;)
    {
        /* Check if 'Enter' key was pressed */
        if (cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1) == CY_RSLT_SUCCESS)
        {
            if (uart_read_value == '\r')
            {
                /* Check the status of CSG DAC */
                if(!Cy_HPPASS_DAC_IsBusy(USER_CSG_SLICE_IDX))
                {
                    user_csg_dac_is_started = false;
                    /* Set DAC ouput buffer value */
                    user_csg_dac_value = USER_CSG_DAC_OUT_HIGH_THRESHOLD;
                    Cy_HPPASS_DAC_SetValue(USER_CSG_SLICE_IDX, user_csg_dac_value);
                    /* Start user CSG DAC HW mode */
                    printf("\r\nStart CSG DAC HW mode\r\n");
                    Cy_HPPASS_DAC_Start(USER_CSG_SLICE_IDX, CY_HPPASS_DAC_HW);
                }
                else
                {
                    /* Stop user CSG DAC HW mode */
                    printf("Stop CSG DAC HW mode\r\n");
                    Cy_HPPASS_DAC_Stop(USER_CSG_SLICE_IDX);
                }
            }
        }

        /* Check if the DAC is started */
        if(user_csg_dac_is_started)
        {
            printf("CSG DAC HW mode is started\r\n");
            user_csg_dac_is_started = false;
        }

        /* Check if the DAC is started and buffer is empty */
        if(user_csg_dac_buf_is_empty && Cy_HPPASS_DAC_IsBusy(USER_CSG_SLICE_IDX))
        {
            user_csg_dac_buf_is_empty = false;
            /* Check current value of DAC output */
            if(USER_CSG_DAC_OUT_HIGH_THRESHOLD <= user_csg_dac_value)
```

```
            {
                user_csg_dac_value = USER_CSG_DAC_OUT_LOW_THRESHOLD;
                printf("Set DAC buffer to 0.8V, current comparator status: %d\r\n", \
                        Cy_HPPASS_Comp_GetStatus(USER_CSG_SLICE_IDX));
            }
            else
            {
                user_csg_dac_value = USER_CSG_DAC_OUT_HIGH_THRESHOLD;
                printf("Set DAC buffer to 2.0V, current comparator status: %d\r\n", \
                        Cy_HPPASS_Comp_GetStatus(USER_CSG_SLICE_IDX));
            }
            /* Set DAC output value */
            Cy_HPPASS_DAC_SetValue(USER_CSG_SLICE_IDX, user_csg_dac_value);
        }
    }
}
```

3.2 Wakeup from Deep Sleep mode using a low-power comparator

The following example demonstrates the voltage comparison functionality of wakeup from Deep Sleep mode using the low-power comparator resource in PSOC™ Control C3 MCU. It configures the low-power comparator (LPCOMP) resource for comparing the internal reference voltage and external voltage input from a dedicated GPIO to wake up the CPU from Deep Sleep mode. An LED is used to indicate the comparison result.

***Note**: The internal reference voltage can vary from 0.4 V to 0.8 V, so the positive input should be greater than 0.8 V to cause a wakeup.*

The block diagram of this example is shown in Figure 10. The example uses low-power comparator 0 to compare the external voltage input from pin P8.0 with the internal reference of LPCOMP. The comparator output is used to wake up the CPU from Deep Sleep mode. It is also connected to the user LED (pin P8.4) by triggering multiplexer group 2 to show the comparator level status.

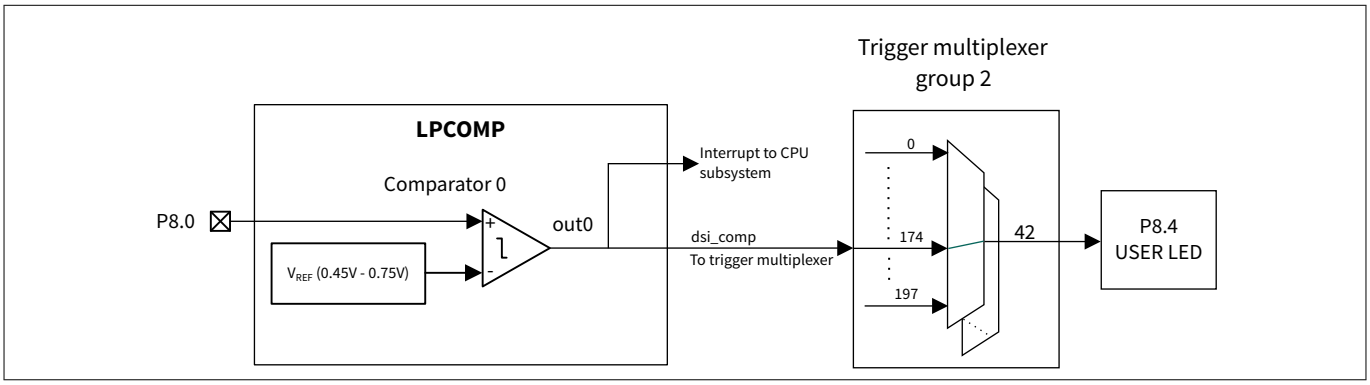

Figure 10 Low-power comparator example

Figure 11 shows the configuration of low-power comparator 0 in the Device Configurator of ModusToolbox™. It does the following:

- Enables the local reference for the comparator negative input
- Sets the positive input to P8.0
- Configures the interrupt to rising edge
- Configures the Power mode of LPCOMP to ultra-low-power for wake up from Deep Sleep mode

3 Comparator configuration and examples

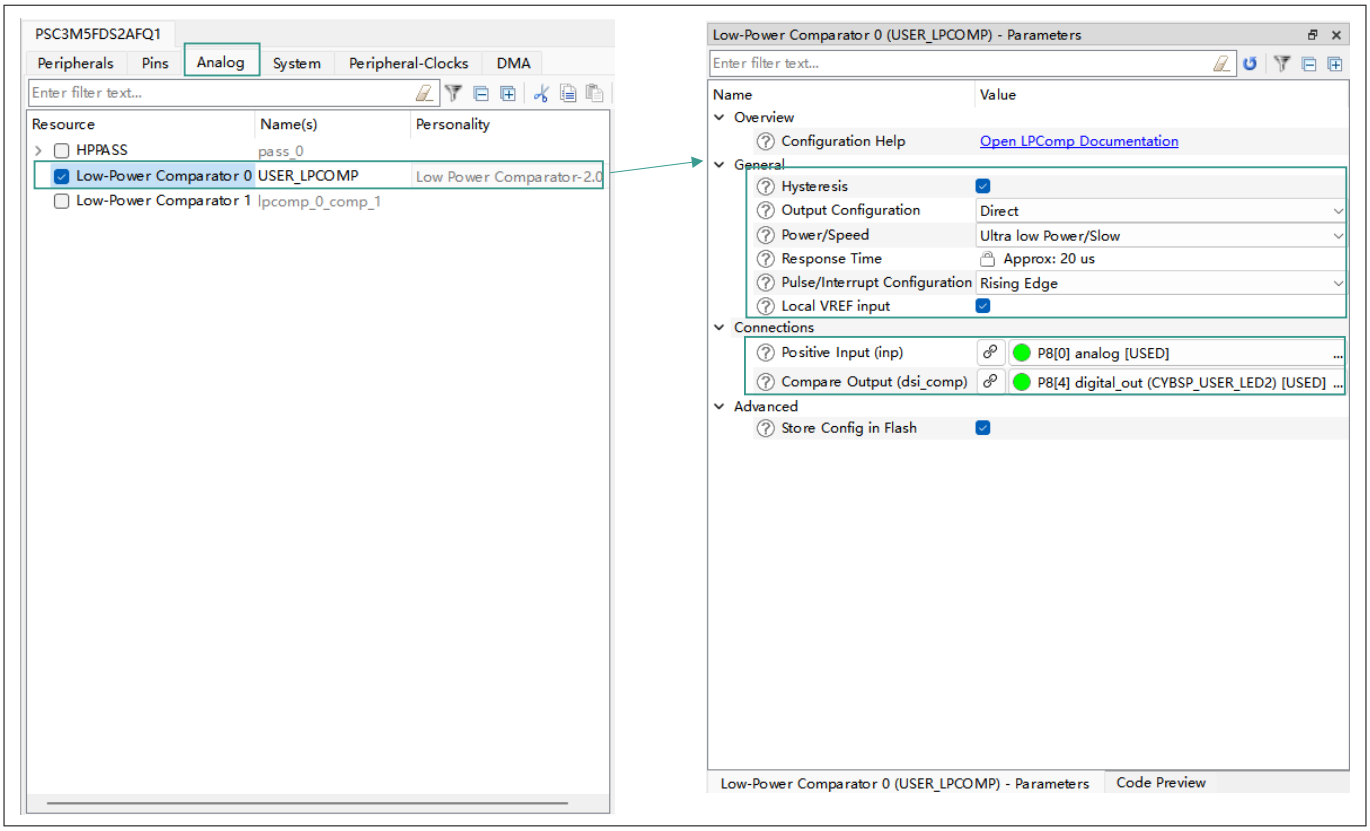

Figure 11 Low-power comparator configuration in ModusToolbox™

The following code snippet shows the source code of this example.

3 Comparator configuration and examples

Source code of wakeup from Deep Sleep mode using LPCOMP

```
#include "cyhal.h"
#include "cybsp.h"
#include "cy_pdl.h"
#include "cy_retarget_io.h"

/* The low-power comparator context */
cy_stc_lpcomp_context_t user_lpcomp_context;

/* The user LPCOMP interrupt configuration structure */
cy_stc_sysint_t user_lpcomp_intr_config =
{
    .intrSrc = USER_LPCOMP_IRQ,
    .intrPriority = 0U,
};

/* This function is the user LPCOMP interrupt handler */
void user_lpcomp_intr_handler(void)
{
    uint32_t intrStatus = Cy_LPComp_GetInterruptStatusMasked(USER_LPCOMP_HW);
    /* Clear interrupt */
    Cy_LPComp_ClearInterrupt(USER_LPCOMP_HW, intrStatus);
}

/* This is the main function for CPU */
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init_fc(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
            CYBSP_DEBUG_UART_CTS,CYBSP_DEBUG_UART_RTS,CY_RETARGET_IO_BAUDRATE);
    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize LPCOMP */
    if (CY_LPCOMP_SUCCESS != Cy_LPComp_Init_Ext(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL,
&USER_LPCOMP_config, &user_lpcomp_context))
    {
        CY_ASSERT(0);
    }
```

```
    /* Configure LPCOMP interrupt */
    Cy_LPComp_SetInterruptMask(USER_LPCOMP_HW, CY_LPCOMP_COMP0);
    Cy_SysInt_Init(&user_lpcomp_intr_config, user_lpcomp_intr_handler);
    NVIC_EnableIRQ(user_lpcomp_intr_config.intrSrc);

    /* Enable LPCOMP */
    Cy_LPComp_Enable_Ext(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL, &user_lpcomp_context);

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
    printf("PSoC Contorl C3 MCU: Wakeup from DeepSleep using a low-power comparator\r\n");

    /* Enable global interrupts */
    __enable_irq();

    for (;;)
    {
        /* If the comparison result is low, goes to the deepsleep mode */
        if(0 == Cy_LPComp_GetCompare(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL))
        {
            printf("Go to DeepSleep mode.\r\n");
            /* Wait for UART traffic to stop */
            while (cy_retarget_io_is_tx_active()) {};

            /* Sets executing CPU to the Deep Sleep mode */
            if(CY_SYSPM_SUCCESS != Cy_SysPm_CpuEnterDeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT))
            {
                printf("The CPU did not enter DeepSleep mode\r\n\n");
                CY_ASSERT(0);
            }
            printf("Wake up from DeepSleep mode.\r\n");
        }
        Cy_SysLib_Delay(5000u);
    }
}
```

3.3 Wakeup from Hibernate mode using a low-power comparator

This section introduces an example that demonstrates the functionality of wakeup from Hibernate mode using a low-power comparator (LPCOMP) in PSOC™ Control C3 MCU. It uses a dedicated GPIO input to compare the input voltage to an internal reference voltage to wake the MCU from Hibernate mode. A user LED indicates the current Power mode.

Figure 12 shows the configuration of low-power comparator 0 in the Device Configurator in ModusToolbox™. It enables the local reference for comparator negative input, and sets positive input to P8.0.

3 Comparator configuration and examples

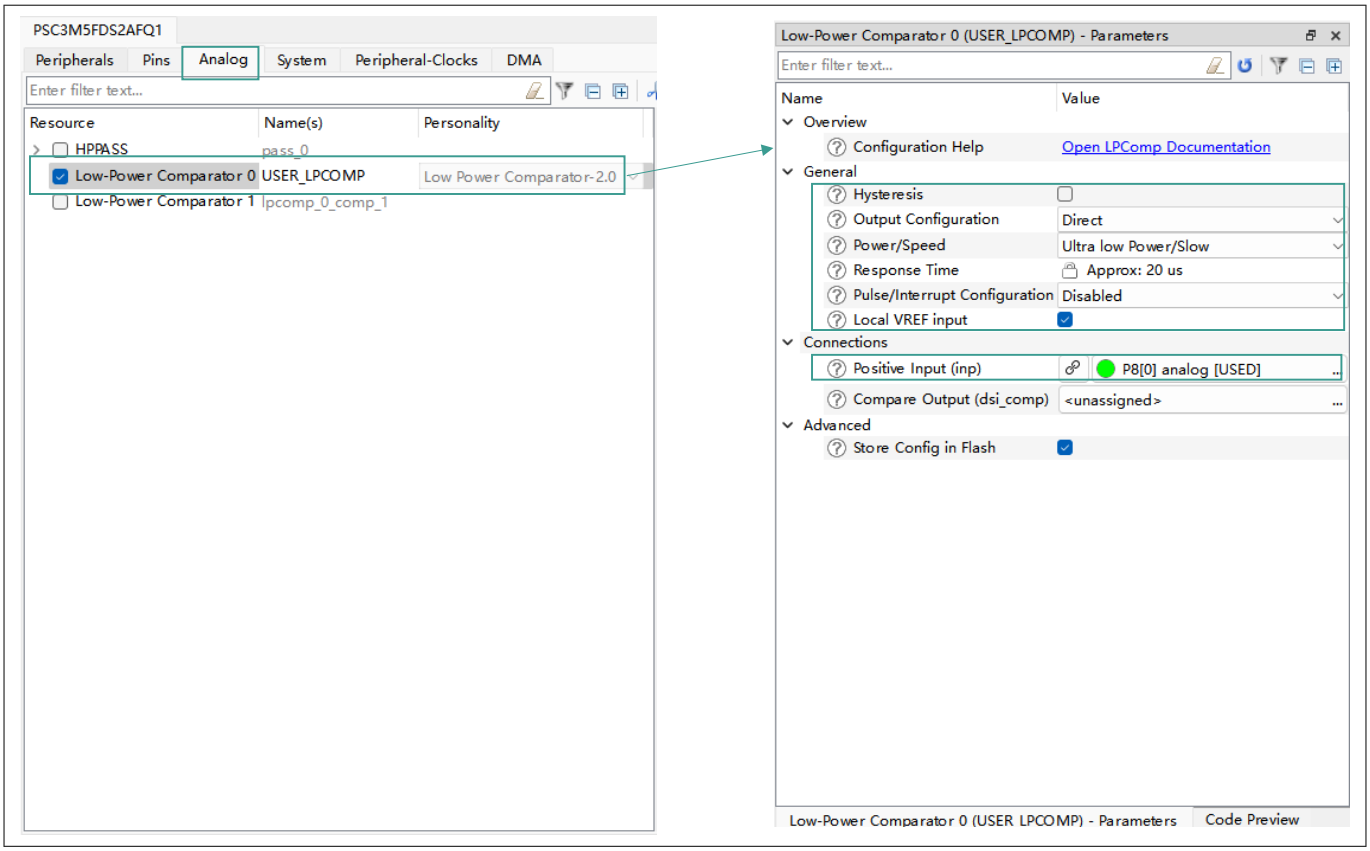

Figure 12 Low-power comparator configuration in ModusToolbox™

Figure 13 shows the firmware flow. The main loop checks the output of the low-power comparator 0 and toggles the LED when the output is HIGH. Otherwise, the system goes into Hibernate mode after turning the LED ON for 2 seconds. The system will wake up immediately if the low-power comparator 0 output goes high during Hibernate mode.

3 Comparator configuration and examples

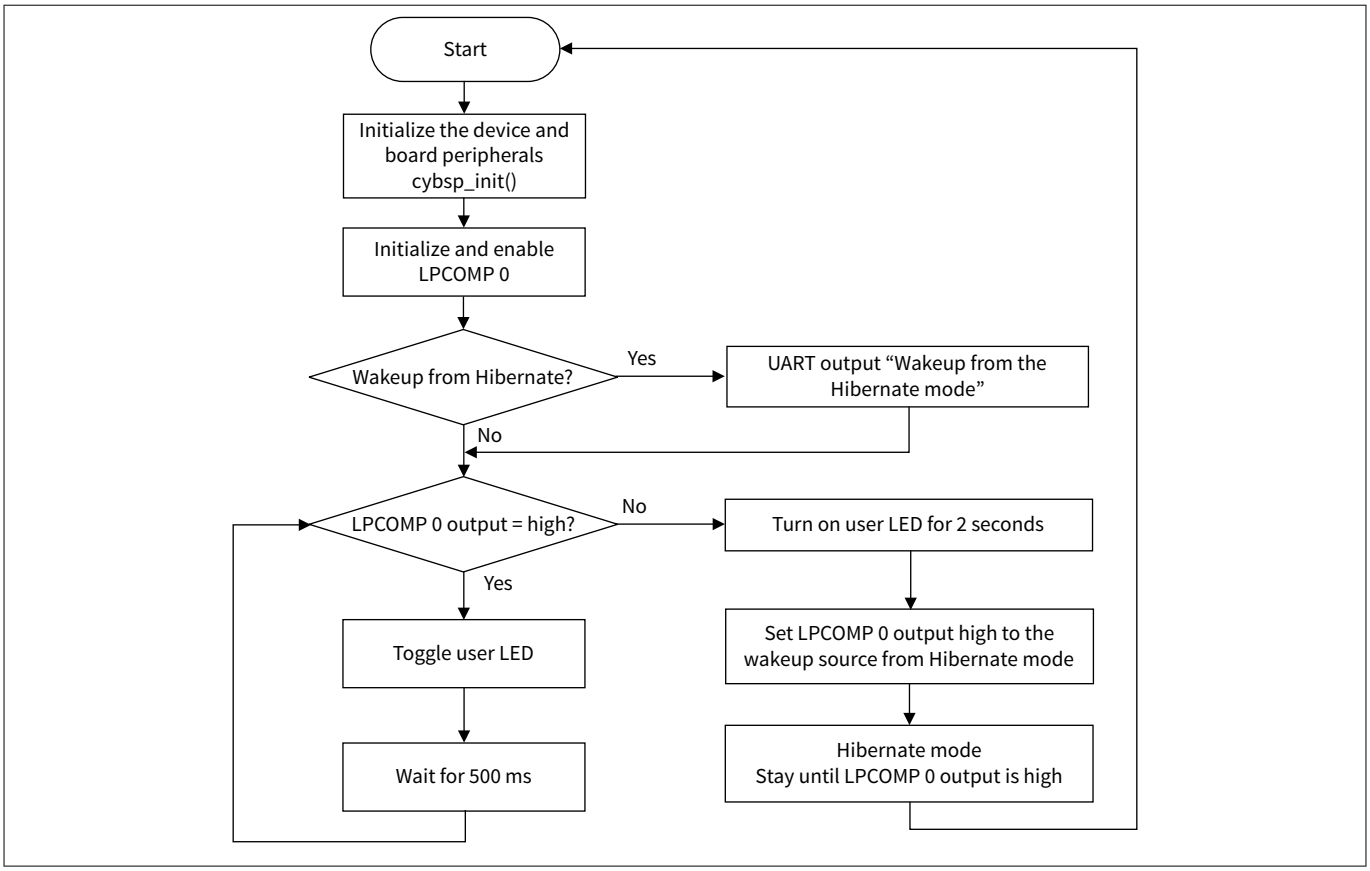

Figure 13 Firmware flow

The following code snippet shows the source code of the example that wake up from Hibernate mode using a low-power comparator.

3 Comparator configuration and examples

Source code of wakeup from Hibernate mode using LPCOMP

```
#include "cyhal.h"
#include "cybsp.h"
#include "cy_pdl.h"
#include "cy_retarget_io.h"

/* The low-power comparator context */
cy_stc_lpcomp_context_t user_lpcomp_context;

/* This is the main function for CPU */
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Initialize retarget-io to use the debug UART port */
    result = cy_retarget_io_init_fc(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
            CYBSP_DEBUG_UART_CTS,CYBSP_DEBUG_UART_RTS,CY_RETARGET_IO_BAUDRATE);
    /* retarget-io init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
    printf("PSoC Contorl C3 MCU: Wakeup from Hibernate using a low-power comparator\r\n");

    /* Check the reset reason */
    if(CY_SYSLIB_RESET_HIB_WAKEUP == (Cy_SysLib_GetResetReason() & CY_SYSLIB_RESET_HIB_WAKEUP))
    {
        /* The reset has occurred on a wakeup from Hibernate power mode */
        printf("Wakeup from the Hibernate mode\r\n");
    }

    /* Initialize LPCOMP */
    if (CY_LPCOMP_SUCCESS != Cy_LPComp_Init_Ext(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL,
&USER_LPCOMP_config, &user_lpcomp_context))
    {
        CY_ASSERT(0);
    }

    /* Enable LPCOMP */
    Cy_LPComp_Enable_Ext(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL, &user_lpcomp_context);
```

3 Comparator configuration and examples

```
    /* Enable global interrupts */
    __enable_irq();

    for (;;)
    {
        /* If the comparison result is high, toggles User LED every 500ms */
        if(0 != Cy_LPComp_GetCompare(USER_LPCOMP_HW, USER_LPCOMP_CHANNEL))
        {
            /* Toggle User LED every 500ms */
            Cy_GPIO_Inv(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM);
            Cy_SysLib_Delay(500u);
            printf("In CPU Active mode, blinking USER LED1 at 500ms \r\n");
        }
        else
        {
            /* Wait for UART traffic to stop */
            while (cy_retarget_io_is_tx_active()) {};
            /* Turn on USER LED for 2 seconds to indicate the MCU entering Hibernate mode. */
            Cy_GPIO_Write(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM, CYBSP_LED_STATE_ON);
            Cy_SysLib_Delay(2000u);
            Cy_GPIO_Write(CYBSP_USER_LED_PORT, CYBSP_USER_LED_NUM, CYBSP_LED_STATE_OFF);
            printf("Turn on the USER LED for 2 seconds, de-initialize IO, and enter System
Hibernate mode. \r\n\n");
            /*Release the UART interface allowing it to be used for other purposes*/
            cy_retarget_io_deinit();

            /* Set the low-power comparator as a wake-up source from Hibernate and jump into
Hibernate */
            Cy_SysPm_SetHibernateWakeupSource(CY_SYSPM_HIBERNATE_LPCOMP0_HIGH);
            if(CY_SYSPM_SUCCESS != Cy_SysPm_SystemEnterHibernate())
            {
                printf("The CPU did not enter Hibernate mode\r\n\n");
                CY_ASSERT(0);
            }
        }
    }
}
```

References

The datasheets and references manuals of PSOC™ Control C3 family are listed here.
Contact Technical Support to obtain these documents.

[1] Datasheets
- PSOC™ Control C3 - PSC3P5xD, PSC3M5xD datasheet
- PSOC™ Control C3 - PSC3P2xD, PSC3M3xD datasheet

[2] Reference manuals
- PSOC™ Control C3 MCU architecture reference manual
- PSOC™ Control C3 MCU registers reference manual

[3] Application notes and user guides
- AN238329 - Getting started with PSOC™ Control C3 MCU on ModusToolbox™ software
- PSOC™ Control C3 MCU hardware design guide
- KIT_PSC3M5_EVK PSOC™ Control C3 Evaluation Kit guide

Revision history

Document revision	Date	Description of changes
**	2024-05-30	Initial release.
*A	2024-12-03	Changed distribution from restricted - NDA required to public.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2024-12-03

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2024 Infineon Technologies AG

All Rights Reserved.

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

Email: erratum@infineon.com

Document reference
IFX-rhu1716781089148

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.

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.