GPIO usage setup in XMC7000 family

About this document

Scope and purpose

AN234118 explains how to use GPIO pins effectively in XMC7000 family MCUs. This application note also explains GPIO basics, configuration options, interrupts, and low-power behavior.

Associated part family

XMC7000 family of XMC™ industrial microcontrollers.

Table of contents

About this document **1**
Table of contents **2**
1 Introduction **3**
2 GPIO pin basics **4**
2.1 Physical structure of GPIO pins 4
2.2 Startup and low-power behavior 6
2.3 Interrupt 6
2.4 Configuration of GPIO output data 7
3 GPIO settings **8**
3.1 Initializing the GPIO 8
3.2 Toggling a port level 9
3.3 Reading an input 10
3.4 Interrupts 11
3.5 Peripheral functions 14
3.5.1 SCB port configuration 14
3.5.2 TCPWM port configuration 22
3.5.3 Analog port configuration 27
4 Glossary **30**
References **31**
Revision history **32**

1 Introduction

XMC7000 family MCUs have a flexible general-purpose I/O (GPIO) architecture that provides additional features compared to traditional MCUs. XMC7000 GPIOs are controlled not only by configuring the registers in firmware, similar to traditional MCUs, but are also driven by custom digital logic and analog block signals. This application note explains the basics of XMC7000 GPIO pins and shows techniques for using them effectively for different functions. To understand more details about the functionality and terminology used in this application note, see the “I/O System” chapter of the XMC7000 architecture technical reference manual (TRM) [2].

2 GPIO pin basics

XMC7000 GPIO pins offer the following features:

- Analog and digital input and output capabilities
- Various drive modes
- Separate port read and write registers
- Slew rate control
- High-speed I/O matrix (HSIOM)
- Edge-triggered interrupts on rising edge, falling edge, or on both the edges, on all GPIO
- Hold mode for latching previous state (used to retain the I/O state in DeepSleep mode)
- Selectable CMOS, TTL, and automotive input buffer mode

The GPIO functionality depends on the peripherals available in XMC7000 family MCUs.

HSIOM is a set of high-speed multiplexers that route internal CPU and peripheral signals to and from GPIOs. HSIOM allows GPIOs to be shared with multiple functions and multiplexes the pin connection to a peripheral that you select. For details on HSIOM connection, see the architecture TRM [2].

2.1 Physical structure of GPIO pins

Figure 1 shows the pin connections with the resources in XMC7000 devices.

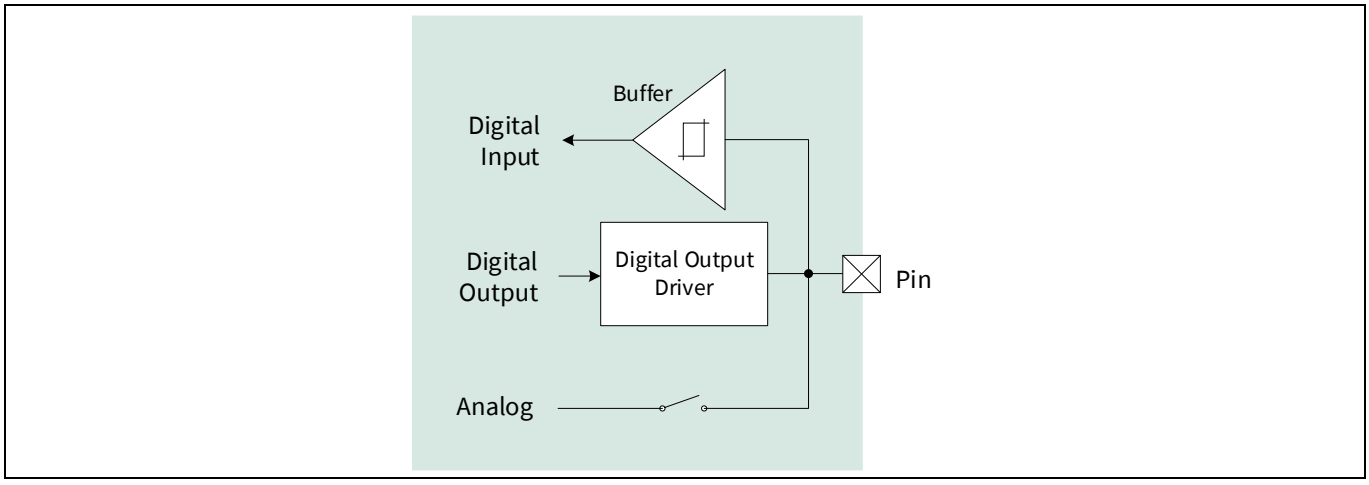

Figure 1 Simplified GPIO block diagram

A detailed block diagram of the GPIO structure is available in the “I/O System” chapter of the architecture TRM [2]. Each pin can act as an input or an output to the digital peripheral such as TCPWM and SCB. Some of the device-specific GPIO can also act as an analog pin for use with ADC.

For more details on the analog pins, see the device datasheet [1]. TCPWM provides timers, counters, pulse width modulators, and SCB provides serial communication functions. See the architecture TRM [2] for details of peripheral functions.

At any given time, you can use a pin for digital input, digital output, analog pin, or even combinations of these three. For example, if you enable both digital output and input, it provides a digital bidirectional pin. The input buffer provides high impedance to the external input, which can be configured as CMOS or TTL.

GPIO pin basics

Figure 2 provides details of the digital output driver in Figure 1. The digital output from each peripheral drives the pin with a digital output driver. The digital output driver supports different drive modes and slew rate control.

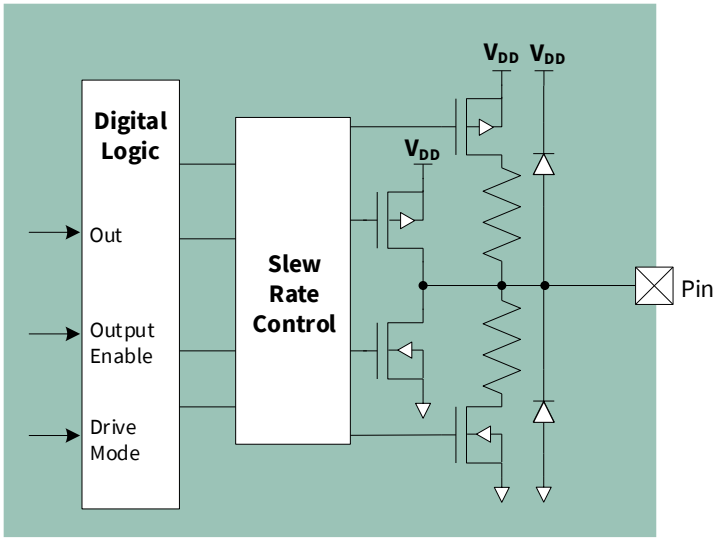

Figure 2 Digital output driver

Slew rate control is provided to reduce the EMI and crosstalk and is configured using the SLOW bit of the Port Output Configuration register (GPIO_PRTx_CFG_OUT). There are two options: Fast and Slow. Slew rate is set to 'Fast' by default. Use the 'Slow' option when the signals are not speed-critical.

XMC7000 device supports various drive modes, which can be configured with the DRIVE_MODE field in the Port Configuration (GPIO_PRTx_CFG) register. Table 1 lists the supported drive modes, and the DRIVE_MODE field setting value. See the architecture TRM [2] for output driver block diagram corresponding to drive modes.

Table 1 Drive modes and applications

DRIVE_MODE [2:0] Field	Drive mode	Application examples
0	High Impedance	Interface to digital input. For digital signals, the input buffer is enabled. For analog signals, the input buffer is typically disabled to reduce the crowbar current and leakage in low-power designs.
2	Resistive Pull-Up	Interface to open-drain LOW input, such as the tachometer output from motors or a switch connected to ground. Pins can be used for either digital input or digital output.
3	Resistive Pull-Down	Interface to an open-drain HIGH input or a switch connected to VDD. Pins can be used for either digital input or digital output.
4	Open Drain, Drives Low	Provides high impedance in the HIGH state and a strong drive in the LOW state; this configuration is used for I^2C pins. This mode works in conjunction with an external pull-up resistor.
5	Open Drain, Drives High	Provides strong drive in the HIGH state and high impedance in the LOW state. This mode works in conjunction with an external pull-down resistor.
6	Strong	CMOS output drives in both LOW and HIGH states

DRIVE_MODE [2:0] Field	Drive mode	Application examples
7	Resistive Pull-Up and Down	Adds a series resistor in both HIGH and LOW states

2.2 Startup and low-power behavior

On reset/power-up, all GPIO pins start up in the high-impedance analog state, that is, with the input buffer and output driver disabled. These GPIO pins remain in this mode until the reset is released. During runtime, GPIOs can be configured by writing to the associated registers.

In Sleep mode, GPIO pins are active and can be actively driven by the peripherals, TCPWM and SCB; only the CPU is inactive in this mode. In DeepSleep mode, the GPIO pins connected to DeepSleep domain peripherals are functional.

XMC7000 MCUs have an additional feature that freezes the GPIOs in DeepSleep and Hibernate modes. The freezing of the GPIO is automatically released when the device comes out of the low power mode. However, note that the GPIOs driven by DeepSleep peripherals are active in DeepSleep mode and are not frozen.

In the case of Hibernate mode, a device reset wakes up the device. This clears the GPIO configuration and pin state and initializes the GPIO pins to high-impedance analog state. Therefore, GPIO reconfiguration is required.

2.3 Interrupt

All port pins have the capability to generate GPIO interrupts. Figure 3 shows GPIO interrupt input block diagram.

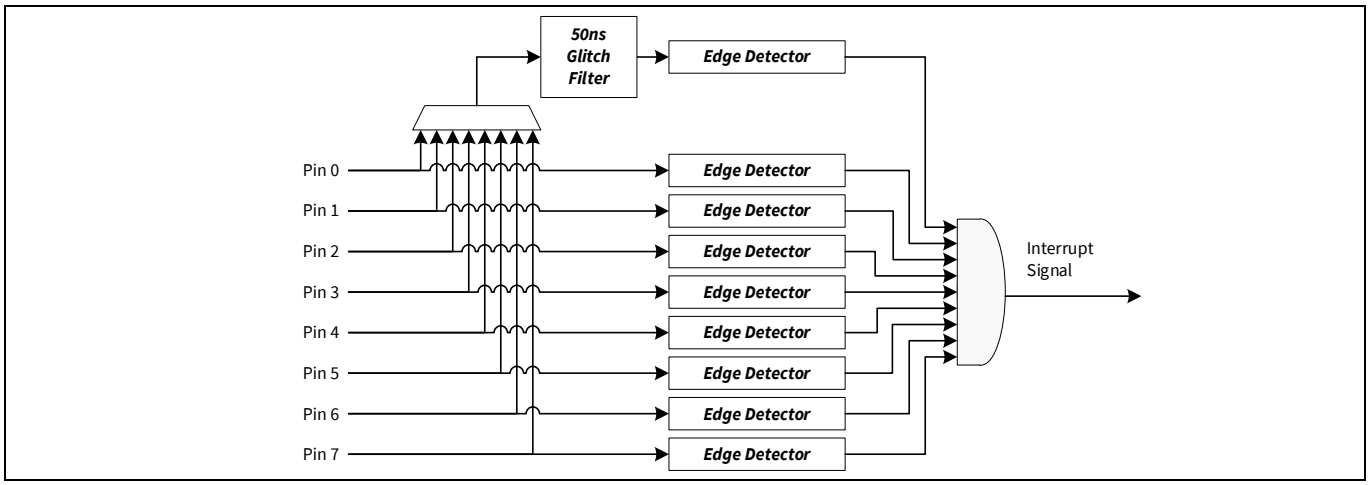

Figure 3 GPIO edge detect block architecture

One GPIO interrupt signal is generated per port. An edge detector is present at each input pin and glitch filter output. The edge detector can detect any of rising-edge, falling-edge, and both edges in the incoming GPIO signal. The glitch filter can be used by one of the pins of a port at a time. Edge detection type is configured by the EDGEx_SEL field in the Port Interrupt Configuration (`GPIO_PRTx_INTR_CFG`) register. Table 2 lists the supported edge detection types and EDGEx_SEL field values. See the architecture TRM [2] for the output driver block diagram corresponding to drive modes.

Table 2 Edge detection setting

EDGEx_SEL field	Edge type
0	Disable
1	Rising edge
2	Falling edge
3	Both edges

Individual GPIO interrupt signals within a port are ORed together to generate a single interrupt request. Thus, there is one interrupt vector for each port. To determine the port that triggered the interrupt, the GPIO_INTR_CAUSEx registers can be read. The software can read this register to determine the pin(s) or glitch filter signal that caused interrupt activation. The software needs to clear the interrupt cause flags to deactivate the interrupt in the interrupt service routine (ISR).

All I/O pins can be used as wakeup interrupts in Sleep and DeepSleep power modes.

2.4 Configuration of GPIO output data

Two types of configurations are available for changing GPIO output data when GPIO is used as output port:

- GPIO_PRTx_OUT
  The write register changes the output data to the written data value, and the read reflects the output data setting. Note that the register read does not reflect the current input of the I/O pins. You need to use read-modify-write to retain the output data of other pins.
- GPIO_PRTx_OUT_CLR and GPIO_PRTx_OUT_SET
  These registers can change the output data in the corresponding I/O pins without affecting the output data of other I/O pins. Writing “1” to the GPIO_PRTx_OUT_CLR register clears the corresponding I/O pin to “0”; writing “0” does not affect the register. Writing “1” to the GPIO_PRTx_OUT_SET register sets the corresponding I/O pin to “1”; writing “0” does not affect the register.

3 GPIO settings

This section provides practical examples on the GPIO pins usage based on the use case with Hardware Abstraction Layer (HAL) provided by Infineon. This section also provides the practical examples of GPIO peripherals function with Peripheral Driver Library (PDL). The use cases portrayed in this document are based on XMC7000 series. For details on the settings of each series, see the datasheets [1] mentioned in References.

The HAL provides a high-level interface to configure and use hardware blocks using ModusToolbox™ software. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all of the low-level peripheral functionality.

The PDL provides the low-level drivers for Infineon devices. The PDL integrates device header files, startup codes, and peripheral drivers into a single package. The drivers abstract the hardware functions into a set of easy-to-use APIs.

3.1 Initializing the GPIO

The HAL library provides *cyhal_gpio.h* and *cyhal_gpio.c* files for the GPIO peripheral usage. It has defined the GPIO events, GPIO direction, GPIO drive modes and others related sources macro. The following parameters must be provided when initializing the GPIO:

- GPIO pin (`cyhal_gpio_t pin`)
- GPIO direction (`cyhal_gpio_direction_t direction`)
- GPIO drive mode (`cyhal_gpio_drive_mode_t drive_mode`)
- GPIO status value (`bool init_val`)

Code Listing 1 demonstrates an example program to initialize the GPIO for writing the port level in the driver part.

Code Listing 1 Example for initializing the GPIO in the HAL driver part

```
 cy_rslt_t cyhal_gpio_init (cyhal_gpio_t pin, cyhal_gpio_direction_t direction, cyhal_gpio_drive_mode_t drive_mode,
bool init_val)
{
    /* Mbed creates GPIOs for pins that are dedicated to other peripherals in some cases. */
#ifndef __MBED__
    cyhal_resource_inst_t pinRsc = _cyhal_utils_get_gpio_resource(pin);
    cy_rslt_t status = cyhal_hwmgr_reserve(&pinRsc);
#else
    cy_rslt_t status = CY_RSLT_SUCCESS;
#endif

    if (status == CY_RSLT_SUCCESS)
    {
        uint32_t pdl_drive_mode = _cyhal_gpio_convert_drive_mode (drive_mode, direction);
        Cy_GPIO_Pin_FastInit (CYHAL_GET_PORTADDR (pin), CYHAL_GET_PIN (pin), pdl_drive_mode, init_val,
HSIOM_SEL_GPIO);
    }

    return status;

}
```

3.2 Toggling a port level

The simple use case of a GPIO is to set the output of a pin to HIGH or LOW in firmware. This example demonstrates the use case of configuring a GPIO pin as an output and toggling the port level in XMC7200 series MCUs. For writing a port level, the drive mode of the GPIO pin must be set to “Strong”.

The CPU alternately sets the pin to “0” or “1” periodically.

Example configuration:

- Port number: CYBSP_USER_LED
- Initial state: High
- Input/Output: Output
- Drive mode: Strong
- Functionality configuration (HSIOM): GPIO
- Interrupt function: Unused
- Slew rate: Fast
- Output driver strength: Strong (Full Drive)

Code Listing 2 demonstrates example program to initialize the GPIO and toggling the port level in the configuration part.

Code Listing 2 Example for initializing the GPIO and toggling the port level in the configuration part

```
int main(void)
{
      cy_rslt_t result;
      uint32_t delay_led_blink = DELAY_LONG_MS;        /*DELAY_LONG_MS = 500*/

    /* Initialize the device and board peripherals */
      result = cybsp_init();

    /* Board init failed. Stop program execution */
      if (result != CY_RSLT_SUCCESS)
       {
           CY_ASSERT(0);
       }

    /* Initialize the user LED */
      result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                               CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

      for(;;)
       {
           /*wait for 0.5s*/
           cyhal_system_delay_ms(delay_led_blink);

           /*toggle led*/
           cyhal_gpio_toggle (CYBSP_USER_LED);
       }
}
```

Code Listing 3 demonstrates an example program to initialize the GPIO for writing the port level in the HAL driver part.

Code Listing 3 Example for toggling the port level in HAL driver part

```
__STATIC_INLINE void cyhal_gpio_toggle_internal (cyhal_gpio_t pin)
{
    Cy_GPIO_Inv (CYHAL_GET_PORTADDR (pin), CYHAL_GET_PIN (pin));
}

#define cyhal_gpio_toggle(pin) cyhal_gpio_toggle_internal(pin)
```

3.3 Reading an input

This example demonstrates a use case for reading from a GPIO pin in XMC7200 series. Enable the digital input buffer for the external digital input and then read the port level. If the GPIO pin is set to read, the drive mode of the pin must be set to “Digital High-Z”.

Example configuration:

- Port number: CYBSP_USER_BTN
- Initial state: High
- Input/Output: Input
- Drive mode: Digital High-Z
- Functionality configuration (HSIOM): 0 (GPIO)
- Interrupt function: Unused

Code Listing 4 demonstrates an example program to read an input in the HAL driver part.

Code Listing 4 Example for reading an input in the configuration part

```
int main(void)
{
    cy_rslt_t result;
    bool value;

    /* 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(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                                        CY_RETARGET_IO_BAUDRATE);

  /* Initialize the user button */
      result = cyhal_gpio_init(CYBSP_USER_BTN, CYHAL_GPIO_DIR_INPUT,
                          CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);
      //CYBSP_BTN_OFF = 1u
      for(;;)
```

Code Listing 4 Example for reading an input in the configuration part

```
        {
                /*Reading out the button pin current value*/
                value = cyhal_gpio_read(CYBSP_USER_BTN);

                /*Output the button pin value with UART*/
                printf("button pin value is %d \r\n", value);
        }
}
```

Code Listing 5 demonstrates an example of reading an input in the HAL driver part.

Code Listing 5 Example of reading an input in the HAL driver part

```
__STATIC_INLINE bool cyhal_gpio_read_internal(cyhal_gpio_t pin)
{
    return 0 != Cy_GPIO_Read(CYHAL_GET_PORTADDR(pin), CYHAL_GET_PIN(pin));
}

#define cyhal_gpio_read(pin) cyhal_gpio_read_internal(pin)
```

3.4 Interrupts

This example demonstrates a use case for interrupt generation from a pin in XMC7200 series. This pin uses one IRQ terminal. Thus, the interrupt source must be identified in the ISR. For the input port, enable the interrupt edge and interrupt mask.

Example configuration:

- Port number: CYBSP_USER_BTN
- Initial state: High
- Input/Output: Input
- Drive mode: Resistive Pull-Up, Input buffer off
- Functionality configuration (HSIOM): GPIO
- Interrupt edge type: Falling edge
- Interrupt function: Used
- Interrupt priority: 7

Code Listing 6 demonstrates an example program to GPIO interrupt generation in the configuration part.

Code Listing 6 Example of GPIO interrupt in the configuration part

```
#define GPIO_INTERRUPT_PRIORITY (7u)

/*******************************************************************************
* Function Prototypes
*******************************************************************************/
static void gpio_interrupt_handler(void *handler_arg, cyhal_gpio_event_t event);

/*******************************************************************************
* Global Variables
*******************************************************************************/
```

GPIO settings

Code Listing 6 Example of GPIO interrupt in the configuration part

```
volatile bool gpio_intr_flag = false;
cyhal_gpio_callback_data_t gpio_btn_callback_data;   // user button call back data


/*******************************************************************************
* Function Name: main
*******************************************************************************/
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 the user LED */
       result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                           CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

      /* Initialize the user button */
       result = cyhal_gpio_init(CYBSP_USER_BTN, CYHAL_GPIO_DIR_INPUT,
                           CYHAL_GPIO_DRIVE_PULLUP, CYBSP_BTN_OFF);

      /* Configure GPIO interrupt */
         gpio_btn_callback_data.callback = gpio_interrupt_handler;

       cyhal_gpio_register_callback(CYBSP_USER_BTN,
                                       &gpio_btn_callback_data);

       cyhal_gpio_enable_event(CYBSP_USER_BTN, CYHAL_GPIO_IRQ_FALL,
                                       GPIO_INTERRUPT_PRIORITY, true);
       /* Enable global interrupts */
       __enable_irq();


          for(;;)
          {
            /*detecting the gpio interrupt*/
             if(gpio_intr_flag == true)
             {
               /*toggle led*/
               cyhal_gpio_toggle(CYBSP_USER_LED);
               gpio_intr_flag = false;
             }
          }
}
```

Figure 4 shows an interrupt handler. When the input port detects an interrupt edge, the interrupt handler is activated. First, read the interrupt status and identify which port pin is being interrupted. Next, clear the interrupt status to detect the next interrupt. Code Listing 7 shows the code example for the interrupt handler.

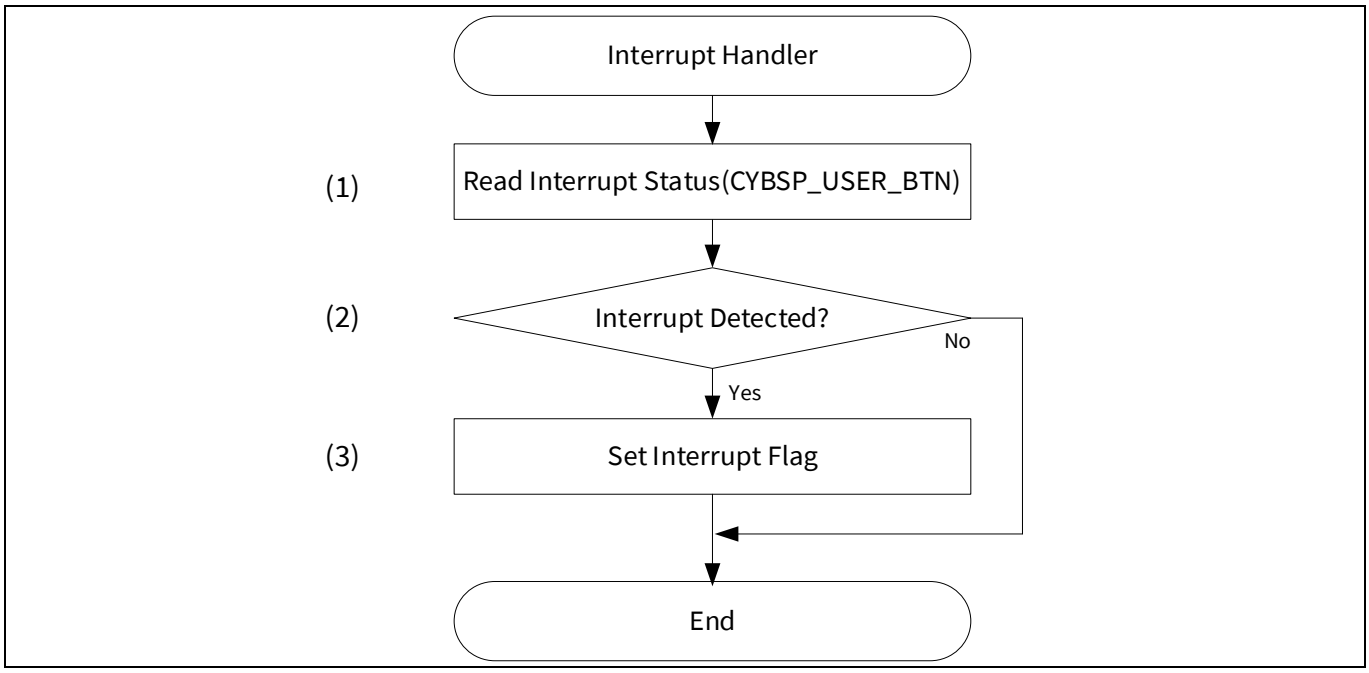

Figure 4 Example of the interrupt handler flow

Code Listing 7 shows an example program for step (3) Set Interrupt Flag of the driver part.

Code Listing 7 Example of the interrupt handler

```
/*******************************************************************************
* Function Name: gpio_interrupt_handler
********************************************************************************
* Summary:
*   GPIO interrupt handler.
*
* Parameters:
*  void *handler_arg (unused)
*  cyhal_gpio_event_t (unused)
*******************************************************************************/
static void gpio_interrupt_handler (void *handler_arg, cyhal_gpio_event_t event)
{
    gpio_intr_flag = true;
}
```

3.5 Peripheral functions

This section explains how to allocate the peripheral functions to the I/O pin. The peripheral function is selected by the HSIOM; the default setting is GPIO. The pin has several specific functions that can be selected.

See [1] for the specific connections available for each pin.

An example port configuration of analog, SCB port, and TCPWM is shown below.

3.5.1 SCB port configuration

This example demonstrates the configuration of a port in XMC7200 series associated with Serial Communication Block (SCB), which is configured as UART. In this example, SCB3 (UART) port (P13.0/P13.1) is used, and it is assigned to ACT_5 of the HSIOM pin connection. Configure the UART function after setting the Rx/Tx port.

For details on selecting an appropriate port for each function, see the *Alternate Function Pin Assignments* section in [1]. For details on UART settings, see AN234127 listed in References.

Figure 5 shows the signal path as an SCB port of P13.0 and P13.1. To use SCB ch3, configure the HSIOM to SCB3_RX/TX.

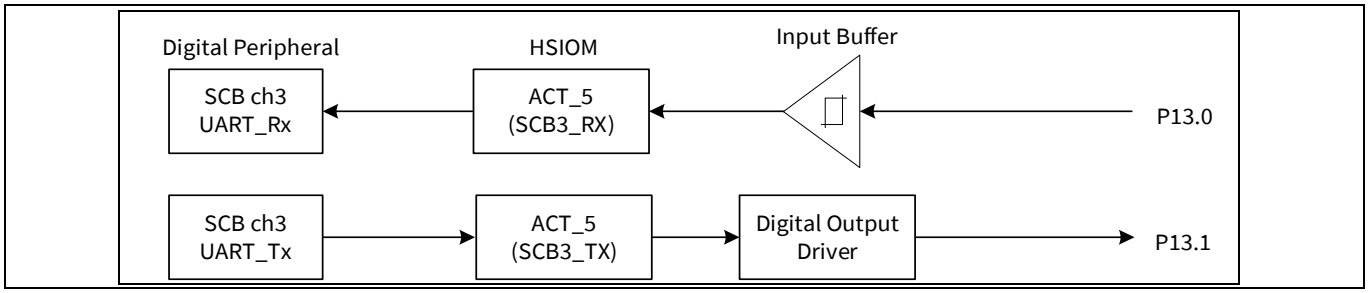

Figure 5 SCB port configuration

Example configuration:

RX port:

- Port number: P13.0
- Initial state: High
- Input/Output: Input
- Drive mode: Digital High-Z
- Functionality configuration (HSIOM): SCB (UART Rx)
- Interrupt function: Unused

TX port:

- Port number: P13.1
- Initial state: High
- Input/Output: Output
- Drive mode: Strong
- Functionality configuration (HSIOM): SCB (UART Tx)
- Interrupt function: Unused

GPIO settings

Table 3 and Table 4 list the parameters of the configuration part in PDL for SCB RX port and TX port, respectively.

Table 3 List of GPIO pin parameters of SCB RX port

Parameters	Description	Value
`outVal`	Selects pin output state.	1ul
`driveMode`	Selects GPIO drive mode for I/O pin. 0: Analog High-Z. Input buffer off 1: Invalid mode. It should not be used 2: Resistive Pull-Up. Input buffer off 3: Resistive Pull-Down. Input buffer off 4: Open Drain, Drives Low. Input buffer off 5: Open Drain, Drives High. Input buffer off 6: Strong Drive. Input buffer off 7: Resistive Pull-Up/Down. Input buffer off 8: Digital High-Z. Input buffer on 9: Invalid mode. It should not be used 10: Resistive Pull-Up. Input buffer on 11: Resistive Pull-Down. Input buffer on 12: Open Drain, Drives Low. Input buffer on 13: Open Drain, Drives High. Input buffer on 14: Strong Drive. Input buffer on 15: Resistive Pull-Up/Down. Input buffer on	CY_GPIO_DM_HIGHZ = 8ul
`.hsiom`	HSIOM selection. Sets connection for I/O pin 0 route.	P13_0_SCB3_UART_RX
`.intEdge`	Sets the edge which will trigger an IRQ for I/O pin 0. 0: Disabled, 1: Rising edge, 2: Falling edge, 3: Both	0ul
`.intMask`	Masks edge interrupt on IO pin 0. 0: Pin interrupt not forwarded to CPU interrupt controller 1: Pin interrupt masked and forwarded to CPU interrupt controller	0ul
`.vtrip`	Selects the pin 0 input buffer voltage trip type. 0: CMOS, 1: TTL	0ul
`.slewRate`	Selects output buffer slew rate for I/O pin 0. 0: Fast slew rate, 1: Slow slew rate	0ul
`.driveSel`	Sets the GPIO drive strength for I/O pin 0. 0: Full drive strength 1: Full drive strength 2: 1/2 drive strength 3: 1/4 drive strength	0ul
`.vregEn`	SIO pair output buffer mode	0ul
`.ibufMode`	SIO pair input buffer mode	0ul
`.vtripSel`	SIO pair input buffer trip point	0ul

GPIO settings

Parameters	Description	Value
.vrefSel	SIO pair reference voltage for input buffer trip point	0ul
.vohSel	SIO pair regulated voltage output level	0ul

Table 4 List of GPIO pin parameters of SCB TX port

Parameters	Description	Value
.outVal	Selects pin output state.	1ul
.driveMode	Selects GPIO drive mode for IO pin. 0: Analog High-Z. Input buffer off 1: Invalid mode. It should not be used 2: Resistive Pull-Up. Input buffer off 3: Resistive Pull-Down. Input buffer off 4: Open Drain, Drives Low. Input buffer off 5: Open Drain, Drives High. Input buffer off 6: Strong Drive. Input buffer off 7: Resistive Pull-Up/Down. Input buffer off 8: Digital High-Z. Input buffer on 9: Invalid mode. It should not be used 10: Resistive Pull-Up. Input buffer on 11: Resistive Pull-Down. Input buffer on 12: Open Drain, Drives Low. Input buffer on 13: Open Drain, Drives High. Input buffer on 14: Strong Drive. Input buffer on 15: Resistive Pull-Up/Down. Input buffer on	CY_GPIO_DM_STRONG_IN_OFF = 6ul
.hsiom	HSIOM selection. Sets connection for I/O pin 1 route.	P13_1_SCB3_UART_TX
.intEdge	Sets the edge which will trigger an IRQ for I/O pin 1. 0: Disabled, 1: Rising edge, 2: Falling edge, 3: Both	0ul
.intMask	Masks edge interrupt on IO pin 1. 0: Pin interrupt not forwarded to CPU interrupt controller 1: Pin interrupt masked and forwarded to CPU interrupt controller	0ul
.vtrip	Selects the pin 1 input buffer voltage trip type. 0: CMOS, 1: TTL	0ul
.slewRate	Selects output buffer slew rate for IO pin 1. 0: Fast slew rate, 1: Slow slew rate	0ul
.driveSel	Sets the GPIO drive strength for IO pin 1. 0: Full drive strength 1: Full drive strength 2: 1/2 drive strength 3: 1/4 drive strength	0ul

GPIO settings

Parameters	Description	Value
.vregEn	SIO pair output buffer mode	0ul
.ibufMode	SIO pair input buffer mode	0ul
.vtripSel	SIO pair input buffer trip point	0ul
.vrefSel	SIO pair reference voltage for input buffer trip point	0ul
.vohSel	SIO pair regulated voltage output level	0ul

The following description will help you understand the configuration of GPIO port, pin, and HSIOM of this example in PDL:

1. Each port number is defined in the device header file. For example, the KIT-XMC72-EVK device header *xmc7200d_e272k8384.h* file in the *mtb_shared\mtb-pdl-cat1\release-v3.0.0\devices\COMPONENT_CAT1C\include* folder.

   ```
   #define GPIO_PRT13     ((GPIO_PRT_Type*) &GPIO->PRT[13])        /* 0x40310680 */
   ```

   The port number and pin number are defined together in the project-specific header in the same folder. For example, the KIT-XMC72-EVK device header file is *gpio_xmc7200_272_bga.h*

   ```
   #define P13_0_PORT              GPIO_PRT13
   #define P13_0_PIN               0u
   #define P13_0_NUM               0u
   #define P13_0_AMUXSEGMENT       AMUXBUS_MAIN

   #define P13_1_PORT              GPIO_PRT13
   #define P13_1_PIN               1u
   #define P13_1_NUM               1u
   #define P13_1_AMUXSEGMENT       AMUXBUS_MAIN
   ```

2. The HSIOM of the specific pin and its parameter value is in the device GPIO header. For example, the KIT-XMC72-EVK device GPIO header file is *gpio_xmc7200_272_bga.h.*

   ```
   /* P13.0 */
   P13_0_GPIO          = 0,                    /* GPIO controls 'out' */
   P13_0_AMUXA         = 4,                    /* Analog mux bus A */
   P13_0_AMUXB         = 5,                    /* Analog mux bus B */
   P13_0_AMUXA_DSI     = 6,                    /* Analog mux bus A, DSI control */
   P13_0_AMUXB_DSI     = 7,                    /* Analog mux bus B, DSI control */
   P13_0_TCPWM1_LINE264    = 8,                /* Digital Active - tcpwm[1].line[264]:0 */
   P13_0_TCPWM1_LINE_COMPL43  = 9,             /* Digital Active - tcpwm[1].line_compl[43]:0 */
   P13_0_TCPWM1_TR_ONE_CNT_IN792 = 10,         /* Digital Active -tcpwm[1].tr_one_cnt_in[792]:0 */
   P13_0_TCPWM1_TR_ONE_CNT_IN130 = 11,         /* Digital Active - tcpwm[1].tr_one_cnt_in[130]:0 */
   P13_0_PASS0_SAR_EXT_MUX_SEL6 = 16,          /* Digital Active - pass[0].sar_ext_mux_sel[6] */
   P13_0_SCB3_UART_RX      = 17,               /* Digital Active - scb[3].uart_rx:0 */
   P13_0_LIN0_LIN_RX3      = 20,               /* Digital Active - lin[0].lin_rx[3]:1 */
   P13_0_SCB3_SPI_MISO     = 21,               /* Digital Active - scb[3].spi_miso:0 */
   P13_0_TCPWM0_TR_ONE_CNT_IN6  = 22,          /* Digital Active - tcpwm[0].tr_one_cnt_in[6] */
   P13_0_AUDIOSS1_MCLK     = 25,               /* Digital Active - audioss[1].mclk:0 */

   /* P13.1 */
   P13_1_GPIO          = 0,                    /* GPIO controls 'out' */
   ```

GPIO settings

```
P13_1_AMUXA          = 4,                /* Analog mux bus A */
P13_1_AMUXB          = 5,                /* Analog mux bus B */
P13_1_AMUXA_DSI      = 6,                /* Analog mux bus A, DSI control */
P13_1_AMUXB_DSI      = 7,                /* Analog mux bus B, DSI control */
P13_1_TCPWM1_LINE44      = 8,            /* Digital Active - tcpwm[1].line[44]:0 */
P13_1_TCPWM1_LINE_COMPL264  = 9,         /* Digital Active - tcpwm[1].line_compl[264]:0 */
P13_1_TCPWM1_TR_ONE_CNT_IN132 = 10,      /* Digital Active - tcpwm[1].tr_one_cnt_in[132]:0 */
P13_1_TCPWM1_TR_ONE_CNT_IN793 = 11,      /* Digital Active - tcpwm[1].tr_one_cnt_in[793]:0 */
P13_1_PASS0_SAR_EXT_MUX_SEL7 = 16,       /* Digital Active - pass[0].sar_ext_mux_sel[7] */
P13_1_SCB3_UART_TX       = 17,           /* Digital Active - scb[3].uart_tx:0 */
P13_1_SCB3_I2C_SDA       = 18,           /* Digital Active - scb[3].i2c_sda:0 */
P13_1_LIN0_LIN_TX3       = 20,           /* Digital Active - lin[0].lin_tx[3]:1 */
P13_1_SCB3_SPI_MOSI      = 21,           /* Digital Active - scb[3].spi_mosi:0 */
P13_1_TCPWM0_LINE_COMPL2    = 22,        /* Digital Active - tcpwm[0].line_compl[2] */
P13_1_AUDIOSS1_TX_SCK    = 25,           /* Digital Active - audioss[1].tx_sck:0 */
```

See the device datasheet for the specific HSIOM functional connections for each pin.

Code Listing 8 demonstrates an example program to initialize the GPIO for SCB RX/TX in the configuration part.

Code Listing 8 Example for initializing GPIO for SCB RX/ TX in the configuration part

```
#define UART_PORT P13_0_PORT
#define UART_RX_NUM P13_0_NUM
#define UART_TX_NUM P13_1_NUM

int main(void)
{
    cy_rslt_t result;
   /* Initialize the device and board peripherals */
   result = cybsp_init();

  /* Enable global interrupts */
  __enable_irq();

  /* Board initialize failed. Stop program execution */
  if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
/* Initialize SCB, Port and Clock */
   Peripheral_Initialization();
}
void Peripheral_Initialization(void)
{
   :
```

GPIO settings

Code Listing 8 Example for initializing GPIO for SCB RX/ TX in the configuration part

```
    /*-----------------------------*/
    /* Port Configuration for UART */
    /*-----------------------------*/

    /* Connect SCB3 UART function to pins */
    Cy_GPIO_SetHSIOM(UART_PORT, UART_RX_NUM, P13_0_SCB3_UART_RX);
    Cy_GPIO_SetHSIOM(UART_PORT, UART_TX_NUM, P13_1_SCB3_UART_TX);

   /* Configure pins for UART operation */
   Cy_GPIO_SetDrivemode(UART_PORT, UART_RX_NUM, CY_GPIO_DM_HIGHZ);
   Cy_GPIO_SetDrivemode(UART_PORT, UART_TX_NUM, CY_GPIO_DM_STRONG_IN_OFF);

   :
}
```

The following show the configuration in ModusToolbox™ Device Configurator:

Figure 6	SCB3 Rx and Tx pins setting
Code Listing 9	SCB3 RX pin configuration PDL code that is auto-generated in GeneratedSource folder
Code Listing 10	SCB3 TX pin configuration PDL code that is auto-generated in GeneratedSource folder
0	SCB3 simple communication function code

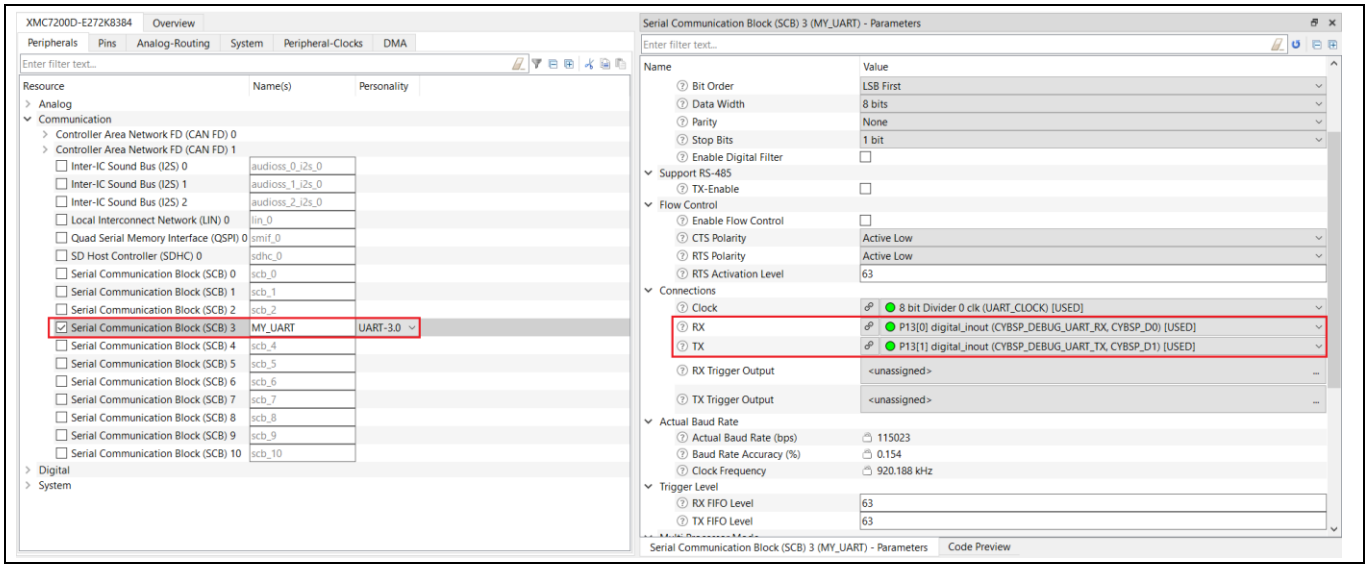

Figure 6 SCB3 pins configuration in Device Configurator

GPIO settings

Code Listing 9 SCB3 RX pin configuration PDL codes in ModusToolbox Device Configurator

```
#define CYBSP_DEBUG_UART_RX_PORT GPIO_PRT13
#define CYBSP_DEBUG_UART_RX_PORT_NUM 13U
#define CYBSP_DEBUG_UART_RX_PIN 0U
#define CYBSP_DEBUG_UART_RX_NUM 0U
#define CYBSP_DEBUG_UART_RX_DRIVEMODE CY_GPIO_DM_HIGHZ
#define CYBSP_DEBUG_UART_RX_INIT_DRIVESTATE 1
#ifndef ioss_0_port_13_pin_0_HSIOM
#define ioss_0_port_13_pin_0_HSIOM HSIOM_SEL_GPIO
#endif
#define CYBSP_DEBUG_UART_RX_HSIOM ioss_0_port_13_pin_0_HSIOM
#define CYBSP_DEBUG_UART_RX_IRQ ioss_interrupts_gpio_13_IRQn
const cy_stc_gpio_pin_config_t CYBSP_DEBUG_UART_RX_config =
{
.outVal = 1,
.driveMode = CY_GPIO_DM_HIGHZ,
.hsiom = CYBSP_DEBUG_UART_RX_HSIOM,
.intEdge = CY_GPIO_INTR_DISABLE,
.intMask = 0UL,
.vtrip = CY_GPIO_VTRIP_CMOS,
.slewRate = CY_GPIO_SLEW_FAST,
.driveSel = CY_GPIO_DRIVE_1_2,
.vregEn = 0UL,
.ibufMode = 0UL,
.vtripSel = 0UL,
.vrefSel = 0UL,
.vohSel = 0UL,
};
void init_cycfg_pins(void)
{
Cy_GPIO_Pin_Init(CYBSP_DEBUG_UART_RX_PORT, CYBSP_DEBUG_UART_RX_PIN,
&CYBSP_DEBUG_UART_RX_config);
}
```

GPIO settings

Code Listing 10 SCB3 TX pin configuration PDL code in Device Configurator

```
#define CYBSP_DEBUG_UART_TX_PORT GPIO_PRT13
#define CYBSP_DEBUG_UART_TX_PORT_NUM 13U
#define CYBSP_DEBUG_UART_TX_PIN 1U
#define CYBSP_DEBUG_UART_TX_NUM 1U
#define CYBSP_DEBUG_UART_TX_DRIVEMODE CY_GPIO_DM_STRONG_IN_OFF
#define CYBSP_DEBUG_UART_TX_INIT_DRIVESTATE 1
#ifndef ioss_0_port_13_pin_0_HSIOM
#define ioss_0_port_13_pin_0_HSIOM HSIOM_SEL_GPIO
#endif
#define CYBSP_DEBUG_UART_TX_HSIOM ioss_0_port_13_pin_0_HSIOM
#define CYBSP_DEBUG_UART_TX_IRQ ioss_interrupts_gpio_13_IRQn

const cy_stc_gpio_pin_config_t CYBSP_DEBUG_UART_TX_config =
{
.outVal = 1,
.driveMode = CY_GPIO_DM_STRONG_IN_OFF,
.hsiom = CYBSP_DEBUG_UART_TX_HSIOM,
.intEdge = CY_GPIO_INTR_DISABLE,
.intMask = 0UL,
.vtrip = CY_GPIO_VTRIP_CMOS,
.slewRate = CY_GPIO_SLEW_FAST,
.driveSel = CY_GPIO_DRIVE_1_2,
.vregEn = 0UL,
.ibufMode = 0UL,
.vtripSel = 0UL,
.vrefSel = 0UL,
.vohSel = 0UL,
};

void init_cycfg_pins(void)
{
Cy_GPIO_Pin_Init(CYBSP_DEBUG_UART_TX_PORT, CYBSP_DEBUG_UART_TX_PIN,
&CYBSP_DEBUG_UART_TX_config);
}
```

Code Listing 11 SCB3 simple communication function code

```
Int main (void)
{
    cy_rslt_t result;

   /* Initialize the device and board peripherals */
       result = cybsp_init();  /*SCB3 RX and TX are initialed within this API*/

    /* Enable global interrupts */
       __enable_irq();

    /* Board initialize failed. Stop program execution */
       if (result != CY_RSLT_SUCCESS)
          {
               CY_ASSERT(0);
          }
     /* Configure UART to operate */
     (void) Cy_SCB_UART_Init(MY_UART, &MY_UART_config, &uartContext);

    /* Enable UART to operate */
     Cy_SCB_UART_Enable(MY_UART);

     /*Put string to uart terminal*/
     Cy_SCB_UART_PutString(MY_UART, "UART Transmit and Receive \r\n");

      for(;;)
      {

      }

}
```

3.5.2 TCPWM port configuration

This example demonstrates the configuration of a port for TCPWM output in XMC7200 series. In this example, TCPWM ch0 port (P6.0) is used, and it is assigned to ACT_0 of HSIOM. Configure the TCPWM function after setting the port.

For details on selecting an appropriate port for each function, see the *Alternate Function Pin Assignments* section in [1]. For details on TCPWM settings, see AN234119 listed in References.

Figure 7 shows the signal path as a TCPWM port on P6.0. To use PWM ch0, configure HSIOM to PWM_0.

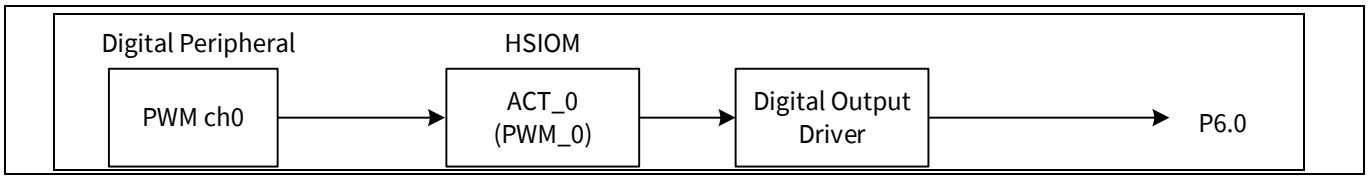

Figure 7 TCPWM port configuration

Example configuration:

- Port number: P6.0
- Initial state: High

GPIO settings

- Input/Output: Output
- Drive mode: Strong
- Functionality configuration (HSIOM): TCPWM
- Interrupt function: Unused

Table 5 lists the parameters of the configuration part in PDL for TCPWM port.

Table 5 GPIO pin parameters of TCPWM port

Parameters	Description	Value
`.outVal`	Selects pin output state.	1ul
`.driveMode`	Selects GPIO drive mode for I/O pin 0: Analog High-Z. Input buffer off 1: Invalid mode. It should not be used 2: Resistive Pull-Up. Input buffer off 3: Resistive Pull-Down. Input buffer off 4: Open Drain, Drives Low. Input buffer off 5: Open Drain, Drives High. Input buffer off 6: Strong Drive. Input buffer off 7: Resistive Pull-Up/Down. Input buffer off 8: Digital High-Z. Input buffer on 9: Invalid mode. It should not be used 10: Resistive Pull-Up. Input buffer on 11: Resistive Pull-Down. Input buffer on 12: Open Drain, Drives Low. Input buffer on 13: Open Drain, Drives High. Input buffer on 14: Strong Drive. Input buffer on 15: Resistive Pull-Up/Down. Input buffer on	CY_GPIO_DM_STRONG_IN_OFF = 6ul
`.hsiom`	HSIOM selection. Sets connection for IO pin 0 route.	P6_0_GPIO/HSIOM_SEL_GPIO
`.intEdge`	Sets the edge which will trigger an IRQ for IO pin 0. 0: Disabled, 1: Rising edge, 2: Falling edge, 3: Both	0ul
`.intMask`	Masks edge interrupt on I/O pin 0. 0: Pin interrupt forwarding disabled 1: Pin interrupt forwarding enabled	0ul
`.vtrip`	Select the pin 0 input buffer mode. 0: CMOS, 1: TTL	0ul
`.slewRate`	Select slew rate for IO pin. 0: Fast slew rate, 1: Slow slew rate	0ul
`.driveSel`	Sets the GPIO drive strength for IO pin 0. 0: Full drive strength 1: Full drive strength 2: 1/2 drive strength 3: 1/4 drive strength	1ul
`.vregEn`	SIO pair output buffer mode	0ul

GPIO settings

Parameters	Description	Value
.ibufMode	SIO pair input buffer mode	0ul
.vtripSel	SIO pair input buffer trip point	0ul
.vrefSel	SIO pair reference voltage for input buffer trip point	0ul
.vohSel	SIO pair regulated voltage output level	0ul

The following description will help you understand the configuration of GPIO port, pin, and HSIOM of this example in PDL:

1. Each port number is defined in the device header file. For example, the KIT-XMC72-EVK device header *xmc7200d_e272k8384.h* file in the *mtb_shared\mtb-pdl-cat1\release-v3.0.0\devices\COMPONENT_CAT1C\include* folder.

   #define GPIO_PRT6 ((GPIO_PRT_Type*) &GPIO->PRT[6])

   The port number and pin number are defined together in the project-specific header in the same folder. For example, the KIT-XMC7C-EVK device header file is *gpio_xmc7200_272_bga.h.*

   #define P6_0_PORT GPIO_PRT6
   #define P6_0_PIN 0u
   #define P6_0_NUM 0u
   #define P6_0_AMUXSEGMENT AMUXBUS_MAIN

2. The HSIOM of the specific pin and its parameter value is in the device GPIO header. For example, the KIT-XMC72-EVK device GPIO header file is *gpio_xmc7200_272_bga.h.*

```
/* P6.0 */
P6_0_GPIO                = 0,    /* GPIO controls 'out' */
P6_0_AMUXA               = 4,    /* Analog mux bus A */
P6_0_AMUXB               = 5,    /* Analog mux bus B */
P6_0_AMUXA_DSI           = 6,    /* Analog mux bus A, DSI control */
P6_0_AMUXB_DSI           = 7,    /* Analog mux bus B, DSI control */
P6_0_TCPWM1_LINE256      = 8,    /* Digital Active - tcpwm[1].line[256]:0 */
P6_0_TCPWM1_LINE_COMPL14 = 9,    /* Digital Active - tcpwm[1].line_compl[14]:0 */
P6_0_TCPWM1_TR_ONE_CNT_IN768  = 10,  /* Digital Active tcpwm[1].tr_one_cnt_in[768]:0 */
P6_0_TCPWM1_TR_ONE_CNT_IN43   = 11,  /* Digital Active - tcpwm[1].tr_one_cnt_in[43]:0 */
P6_0_TCPWM1_TR_ONE_CNT_IN1569 = 16,  /* Digital Active - tcpwm[1].tr_one_cnt_in[1569]:0 */
P6_0_SCB4_UART_RX        = 17,   /* Digital Active - scb[4].uart_rx:0 */
P6_0_SCB4_SPI_MISO       = 19,   /* Digital Active - scb[4].spi_miso:0 */
P6_0_LIN0_LIN_RX3        = 20,   /* Digital Active - lin[0].lin_rx[3]:0 */
P6_0_TCPWM0_LINE0        = 22,   /* Digital Active - tcpwm[0].line[0] */
P6_0_LIN0_LIN_EN9        = 23,   /* Digital Active - lin[0].lin_en[9]:1 */
```

Figure 8 shows the pin assignment with alternate functions of P6.0. PWM1_M_0 indicate the 16-bit TCPWM with motor control line out, TCPWM1 block with counter 0.

P6.0	PWM1_M_0	PWM1_14_N	TC1_M_0_TR0	TC1_14_TR1	TC1_H_11_TR0	SCB4_RX		SCB4_MISO			PWM0_0				
P6.1	PWM1_0	PWM1_M_0_N	TC1_0_TR0	TC1_M_0_TR1	TC1_H_11_TR1	SCB4_TX	SCB4_SDA	SCB4_MOSI							
P6.2	PWM1_M_1	PWM1_0_N	TC1_M_1_TR0	TC1_0_TR1	PWM1_H_12	SCB4_RTS	SCB4_SCL	SCB4_CLK		CAN0_2_TX	PWM0_0_N			SDHC_CARD_MECH_WRITE_PROT	
P6.3	PWM1_1	PWM1_M_1_N	TC1_1_TR0	TC1_M_1_TR1	PWM1_H_12_N	SCB4_CTS		SCB4_SEL0		CAN0_2_RX		SPIHB_CLK		SDHC_CARD_CMD	

Figure 8 P6.0 pin assignment with alternate functions in XMC7200 series

1	PWMx_y	TCPWM	TCPWM 16-bit PWM (no motor control), PWM_DT and PWM_PR line out, x-TCPWM block, y-counter number
2	PWMx_y_N	TCPWM	TCPWM 16-bit PWM (no motor control), PWM_DT and PWM_PR complementary line out (N), x-TCPWM block, y-counter number
3	PWMx_M_y	TCPWM	TCPWM 16-bit PWM with motor control line out, x-TCPWM block, y-counter number
4	PWMx_M_y_N	TCPWM	TCPWM 16-bit PWM with motor control complementary line out (N), x-TCPWM block, y-counter number
5	PWMx_H_y	TCPWM	TCPWM 32-bit PWM, PWM_DT and PWM_PR line out, x-TCPWM block, y-counter number
6	PWMx_H_y_N	TCPWM	TCPWM 32-bit PWM, PWM_DT and PWM_PR complementary line out (N), x-TCPWM block, y-counter number

Figure 9 Pin function description in the datasheet

Figure 10 shows the initialize GPIO for TCPWM in the ModusToolbox Device Configurator Tool.

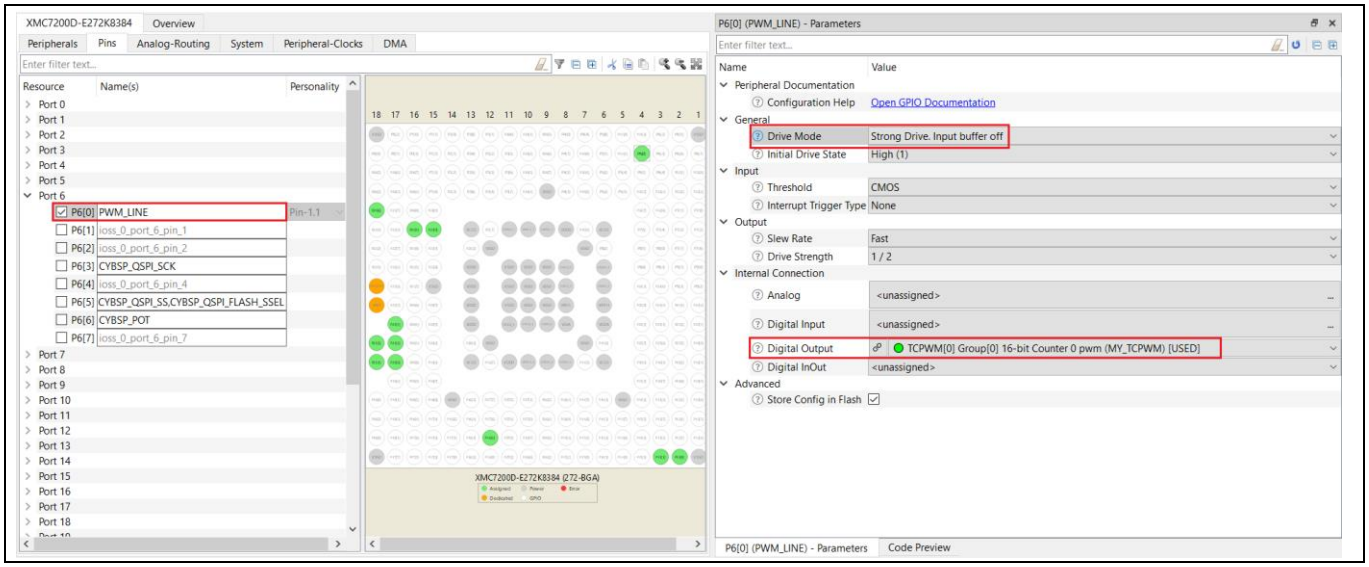

Figure 10 GPIO initial as PWM line output

Code Listing 12 shows the corresponding the PDL GPIO initial code in ModusToolbox™ Device Configurator. And the code is auto-generated in GeneratedSource folder

Code Listing 12 Example for initializing GPIO for TCPWM in Device Configurator

```
#define PWM_LINE_PORT GPIO_PRT6
#define PWM_LINE_PORT_NUM 6U
#define PWM_LINE_PIN 0U
#define PWM_LINE_NUM 0U
#define PWM_LINE_DRIVEMODE CY_GPIO_DM_STRONG_IN_OFF
#define PWM_LINE_INIT_DRIVESTATE 1

#ifndef ioss_0_port_6_pin_0_HSIOM
#define ioss_0_port_6_pin_0_HSIOM HSIOM_SEL_GPIO
#endif

#define PWM_LINE_HSIOM ioss_0_port_6_pin_0_HSIOM

#define PWM_LINE_IRQ ioss_interrupts_gpio_6_IRQn

const cy_stc_gpio_pin_config_t PWM_LINE_config =
{
```

GPIO settings

Code Listing 12 Example for initializing GPIO for TCPWM in Device Configurator

```
.outVal = 1,
.driveMode = CY_GPIO_DM_STRONG_IN_OFF,
.hsiom = PWM_LINE_HSIOM,
.intEdge = CY_GPIO_INTR_DISABLE,
.intMask = 0UL,
.vtrip = CY_GPIO_VTRIP_CMOS,
.slewRate = CY_GPIO_SLEW_FAST,
.driveSel = CY_GPIO_DRIVE_1_2,
.vregEn = 0UL,
.ibufMode = 0UL,
.vtripSel = 0UL,
.vrefSel = 0UL,
.vohSel = 0UL,
};
void init_cycfg_pins(void)
{
  Cy_GPIO_Pin_Init(PWM_LINE_PORT, PWM_LINE_PIN, &PWM_LINE_config);
}
```

Code Listing 13 shows the PWM line output with GPIO P6.0 pin use case.

Code Listing 13 Example for PWM line output with GPIO

```
int main(void)
{
       cy_rslt_t result;
      /* Initialize the device and board peripherals */
      result = cybsp_init();    /*PWM_LINE_PIN initial within this API*/

      /* Enable global interrupts */
      __enable_irq();

      /* Board initialize failed. Stop program execution */
      if (result != CY_RSLT_SUCCESS)
          {
              CY_ASSERT(0);
          }

      if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(MY_TCPWM, MY_TCPWM_NUM,
&MY_TCPWM_config))
          {
             /* Handle possible errors */
          }

       /* Enable the initialized PWM */
       Cy_TCPWM_PWM_Enable(MY_TCPWM, MY_TCPWM_NUM);
       /* Then start the PWM */
       Cy_TCPWM_TriggerStart_Single(MY_TCPWM, MY_TCPWM_NUM);

          for(;;)
          {

          }

}
```

3.5.3 Analog port configuration

This example demonstrates the configuration of an analog port in XMC7200 series. In this example, SAR2 ADC[0]_0 is assigned to an analog input. The GPIO drive mode of the port is set to High-Z and the input buffer is disabled to avoid crowbar current. For details on SAR ADC setting, see AN234282 listed in References.

Example configuration:

- Port number: P16.0
- Initial state: Low
- Input/Output: Input
- Drive mode: High-Z
- Functionality configuration (HSIOM): GPIO
- Interrupt function: Unused

Table 6 lists the parameters of the configuration part in PDL for analog port.

Table 6 List of GPIO pin parameters

Parameters	Description	Value
`.outVal`	Selects pin output state.	0ul
`.driveMode`	Selects GPIO drive mode for I/O pin. 0: Analog High-Z. Input buffer off 1: Invalid mode. It should not be used 2: Resistive Pull-Up. Input buffer off 3: Resistive Pull-Down. Input buffer off 4: Open Drain, Drives Low. Input buffer off 5: Open Drain, Drives High. Input buffer off 6: Strong Drive. Input buffer off 7: Resistive Pull-Up/Down. Input buffer off 8: Digital High-Z. Input buffer on 9: Invalid mode. It should not be used 10: Resistive Pull-Up. Input buffer on 11: Resistive Pull-Down. Input buffer on 12: Open Drain, Drives Low. Input buffer on 13: Open Drain, Drives High. Input buffer on 14: Strong Drive. Input buffer on 15: Resistive Pull-Up/Down. Input buffer on	CY_GPIO_DM_ANALOG = 0ul
`.hsiom`	Sets connection for IO pin 0 route.	P16_0_GPIO/ HSIOM_SEL_GPIO
`.intEdge`	Sets the edge which will trigger an IRQ for IO pin 0. 0: Disabled, 1: Rising edge, 2: Falling edge, 3: Both	0ul
`.intMask`	Masks edge interrupt on IO pin 0. 0: Pin interrupt forwarding disabled 1: Pin interrupt forwarding enabled	0ul
`.vtrip`	Selects the pin 0 input buffer mode.	0ul

GPIO settings

Parameters	Description	Value
	0: CMOS, 1: TTL	
.slewRate	Selects slew rate for IO pin. 0: Fast slew rate, 1: Slow slew rate	0ul
.driveSel	Sets the GPIO drive strength for IO pin. 0: Full drive strength 1: Full drive strength 2: 1/2 drive strength 3: 1/4 drive strength	0ul
.vregEn	SIO pair output buffer mode	0ul
.ibufMode	SIO pair input buffer mode	0ul
.vtripSel	SIO pair input buffer trip point	0ul
.vrefSel	SIO pair reference voltage for input buffer trip point	0ul
.vohSel	SIO pair regulated voltage output level	0ul

The following description will help you understand the configuration of GPIO port, pin, and HSIOM of this example in PDL:

1. Each port number is defined in the device header file. For example, the KIT-XMC72-EVK device header *xmc7200d_e272k8384.h* file in the *mtb_shared\mtb-pdl-cat1\release-v3.0.0\devices\COMPONENT_CAT1C\include* folder.

   ```
   #define GPIO_PRT16    ((GPIO_PRT_Type*) &GPIO->PRT[16])
   ```

   The port number and pin number are defined together in the project-specific header in the same folder. For example, the KIT-XMC7C-EVK device header file is *gpio_xmc7200_272_bga.h.*

   ```
   #define P16_0_PORT              GPIO_PRT16
   #define P16_0_PIN               0u
   #define P16_0_NUM               0u
   #define P16_0_AMUXSEGMENT       AMUXBUS_MAIN
   ```

2. The HSIOM of the specific pin and its parameter value is in the device GPIO header. For example, the KIT-XMC72-EVK device GPIO header file is *gpio_xmc7200_272_bga.h.*

   ```
   /* P16.0 */
   P16_0_GPIO              = 0,                    /* GPIO controls 'out' */
   P16_0_AMUXA             = 4,                    /* Analog mux bus A */
   P16_0_AMUXB             = 5,                    /* Analog mux bus B */
   P16_0_AMUXA_DSI         = 6,                    /* Analog mux bus A, DSI control */
   P16_0_AMUXB_DSI         = 7,                    /* Analog mux bus B, DSI control */
   P16_0_TCPWM1_LINE60     = 8,                    /* Digital Active - tcpwm[1].line[60]:0 */
   P16_0_TCPWM1_LINE_COMPL59   = 9,                /* Digital Active - tcpwm[1].line_compl[59]:0 */
   P16_0_TCPWM1_TR_ONE_CNT_IN180 = 10,             /* Digital Active - tcpwm[1].tr_one_cnt_in[180]:0 */
   P16_0_TCPWM1_TR_ONE_CNT_IN178 = 11,             /* Digital Active - tcpwm[1].tr_one_cnt_in[178]:0 */
   P16_0_TCPWM1_LINE512    = 16,                   /* Digital Active - tcpwm[1].line[512]:1 */
   P16_0_SCB9_SPI_SELECT1  = 19,                   /* Digital Active - scb[9].spi_select1:0 */

   P16_0_LIN0_LIN_RX11     = 20,                   /* Digital Active - lin[0].lin_rx[11]:0 */
   ```

GPIO settings

Code Listing 14 demonstrates an example program to initialize GPIO for analog port in the configuration part.

Code Listing 14 Example for initializing GPIO for analog port in configuration part

```
#define CY_ADC_POT_PORT GPIO_PRT16
#define CY_ADC_POT_PIN   P16_0_PIN
int main(void)
{
    :
    /* ADC port setting (Note default port setting after reset is just fine) */
    {
       cy_stc_gpio_pin_config_t adcPinConfig =
       {
           .outVal    = 0ul,       /* Pin output state */
           .driveMode = CY_GPIO_DM_ANALOG, /* Drive mode */
           .hsiom     = HSIOM_SEL_GPIO, /* HSIOM selection */
           .intEdge   = 0ul,       /* Interrupt Edge type */
           .intMask   = 0ul,       /* Interrupt enable mask */
           .vtrip     = 0ul,       /* Input buffer voltage trip type */
           .slewRate  = 0ul,       /* Output buffer slew rate */
           .driveSel  = 0ul,       /* Drive strength */
       };
       Cy_GPIO_Pin_Init(CY_ADC_POT_PORT, CY_ADC_POT_PIN, &adcPinConfig);
    }
:
}
```

4 Glossary

Table 7 Glossary

Terms	Description
ADC	Analog-to-digital converter. See the “SAR ADC” chapter of the architecture TRM [2] for details.
GPIO	General-purpose I/O
HSIOM	High-speed I/O matrix. See the “High-Speed I/O Matrix” section in the *I/O System* chapter of the architecture TRM [2] for details.
SCB	Serial communication block. See the “Serial Communications Block (SCB)” chapter of the architecture TRM [2] for details.
Slew rate control	The change of voltage per unit of time. See the “Slew Rate Control” section in the *I/O System* chapter of the architecture TRM [2] for details.
TCPWM	Timer, Counter, and Pulse Width Modulator. See the “Timer, Counter, and PWM” chapter of the architecture TRM [2] for details.

References

Contact Technical Support to obtain XMC7000 family reference documents.

[1] Device datasheet

- 002-33896: XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- 002-33522: XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

[2] Device architecture TRM

- 002-33816: XMC7000 MCU family architecture technical reference manual

[3] Application notes

- AN234226 – Interrupt usage in XMC7000 family
- AN234119 – Timer, Counter and PWM(TCPWM) usage in XMC7000 family
- AN234127 – Serial communications block (SCB) usage in XMC7000 family
- AN234282 – Using a SAR ADC in XMC7000 family

Revision history

Document revision	Date	Description of changes
**	2021-11-08	Initial release
*A	2022-04-19	Updated and added the GPIO use cases code snippets
*B	2023-02-27	Updated the broken links. Updated to new template.

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

Edition 2023-02-27

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-34118 Rev. *B

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.