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

1 Introduction

XMC7000 family MCUs have a flexible general-purpose I/O (GPIO) architecture that provides additional features than 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 device.

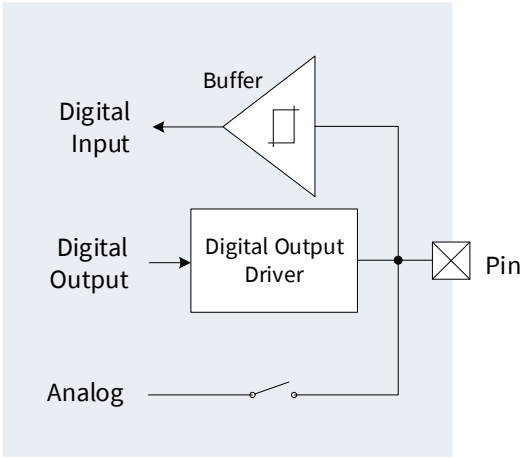

Figure 1 Simplified GPIO block diagram

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 details 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.

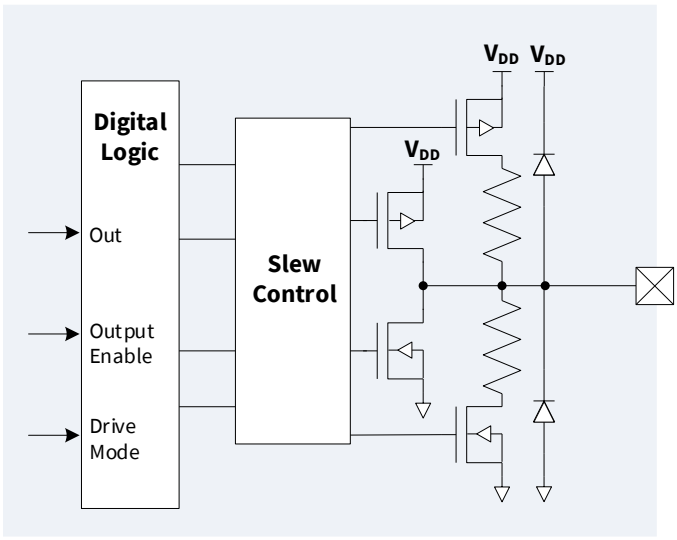

Figure 2 Digital output driver

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
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.

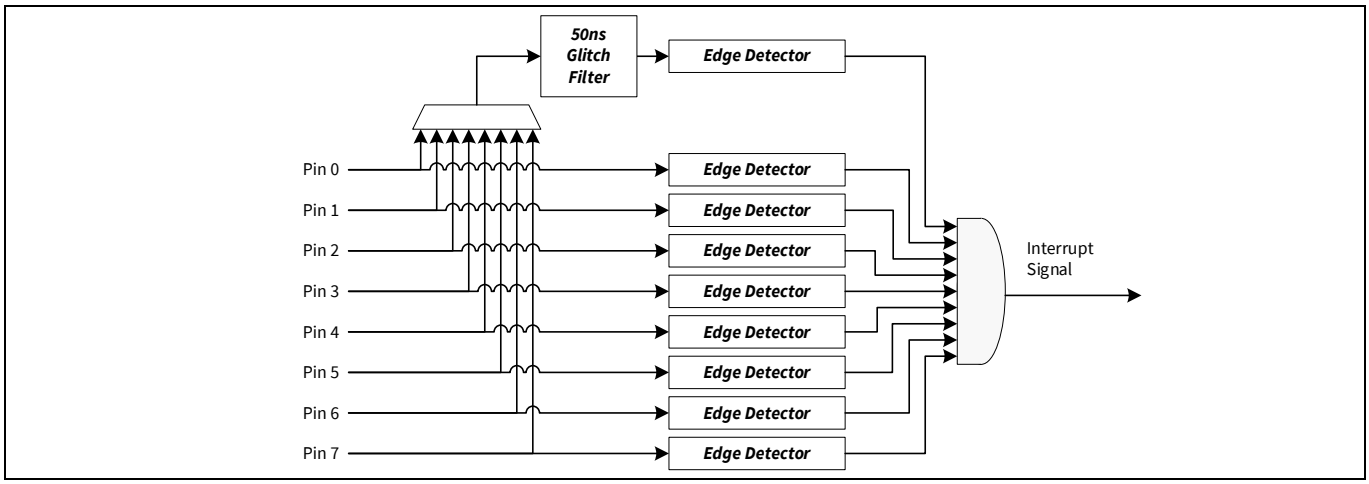

Figure 3 GPIO edge detect block architecture

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

GPIO pin basics

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 mode.

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. 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.

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 GPIO in 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 XMC7000 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 driver part.

Code Listing 3 Example for toggling the port level in the 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 XMC7000 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 Impedance”.

Example configuration:

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

Code Listing 5 demonstrates an example program to read an input in the 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(;;)
        {
                /*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 driver part.

Code Listing 5 Example of reading an input in the 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 Interrupt

This example demonstrates a use case for interrupt generation from a pin in XMC7000 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: Digital High impedance
- 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
*******************************************************************************/

volatile bool gpio_intr_flag = false;

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 */
         cyhal_gpio_register_callback(CYBSP_USER_BTN,
                                      gpio_interrupt_handler, NULL);
         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;
               }
            }

}
```

GPIO settings

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.

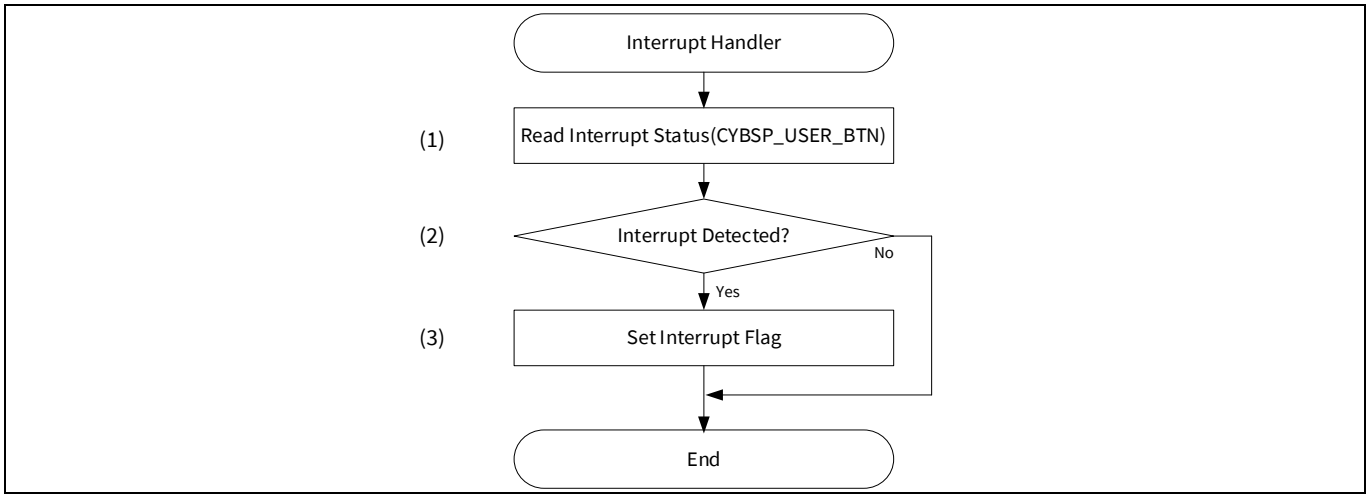

Figure 4 Example of the interrupt handler flow

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_irq_event_t (unused)
*******************************************************************************/
static void gpio_interrupt_handler (void *handler_arg, cyhal_gpio_irq_event_t event)
{
    gpio_intr_flag = true;
}
```

4 Glossary

Table 3 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

The following are the XMC7000 series datasheets, technical reference manuals, and application notes. Contact **Technical Support** to obtain these 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 architecture technical reference manual

[3] Application notes

- AN234226 – How to use interrupt in XMC7000 family

Revision history

Document version	Date of release	Description of changes
**	2021-11-08	Initial release

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

Edition 2021-11-08

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

002-34118 Rev. **

IMPORTANT NOTICE

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

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

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

WARNINGS

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

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