GigaDevice Semiconductor Inc.

GD32M531 Software Development Guide

Application Note

AN317

Revision 1.0

(Mar. 2026)

Table of Contents

List of Figures

List of Tables

1. Introduction

This application note is specifically written for the GD32M531 series MCUs, aiming to provide functional examples and usage precautions for the peripheral resources of this series of chips, helping users quickly master the software development methods and practices for GD32M531 series MCUs. The applicable products are as shown in ***Table 1-1. Applicable product***.

Table 1-1. Applicable product

Product series	Model
GD32M531	GD32M531 series

Note: This application note is for reference only. In case of any conflict with the user manual or datasheet, the user manual or datasheet shall prevail.

2. Module usage instructions

2.1. System and memory architecture(SYSTEM)

2.1.1. Boot configuration

The GD32M531 devices provide two kinds of boot sources which can be selected by the BOOT-PN6. The details are shown in the following table. The value on the two pins is latched on the 4th rising edge of CK_SYS after a reset. It is up to the user to set the BOOT-PN6 pin after a power-on reset or a system reset to select the required boot source. Once the pin has been sampled, it is free and can be used for other purposes.

Table 2-1. Boot modes

Selected boot source	Boot mode selection pin
	BOOT-PN6
System Memory	0
Main Flash Memory	1

After power-on sequence or a system reset, the Arm® Cortex®-M33 processor fetches the top-of-stack value from address 0x0000 0000 and the base address of boot code from 0x0000 0004 in sequence. Then, it starts executing code from the base address of boot code.

According to the selected boot source, either the main flash memory (original memory space beginning at 0x0800 0000) or the system memory (original memory space beginning at 0x1FFF E000) is aliased in the boot memory space which begins at the address 0x0000 0000.

2.1.2. Notes on the SYSCFG Module

After the MCU reset, the registers of the RCC and PMU modules are protected and cannot be modified directly. They must first be unlocked through the SYSCFG_PRCFG register in the SYSCFG module before modification is allowed. The unlocking process is as follows:

1. Write the key value (0xA5) to the PRKEY[7:0] bits of the SYSCFG_PRCFG register to unlock the PRCFG0, PRCFG1, and PRCFG2 registers.
2. Use PRCFG0/1/2 to enable write access.
3. Configure the PMU or RCU registers.

The corresponding PMU and RCU module registers for PRCFG0/1/2 can be found in the SYSCFG_PRCFG register section of the user manual.

2.2. Power management unit (PMU)

2.2.1. Precautions for PMU using LVD1 and LVD2

LVD1 is used to detect whether the VDD / VDDA supply voltage is lower or higher than a programmed threshold selected by the LVD1T[3:0] bits in the PMU_LVD1CTL register. LVD2 is used to detect whether the VDD / VDDA supply voltage is lower or higher than a programmed threshold selected by the LVD2T[3:0] bits in the PMU_LVD2CTL register. Before configuring LVD1 and LVD2, the PRCFG2 bit in the SYSCFG_PRCFG register must be set first to enable writing to the PMU_LVD1CTL and PMU_LVD2CTL registers.

2.3. Reset and clock unit (RCU)

2.3.1. Precautions for the use of RCU

After enabling the HXTAL clock monitor function, an NMI will be generated by default if the HXTAL clock loss occurs. If NMI generation is not required, call syscfg_interrupt_disable(SYSCFG_HXTAL_NIMIE) to disable it.

2.4. Interrupt/event controller (EXTI)

2.4.1. EXTI usage of PN2

1. PN2 can be connected to EXTI0 to trigger interrupts and events, and it can be configured following the standard EXTI pin configuration procedure.

Table 2-2. Example of PN2 connect to EXTI0 configuration

```
/* enable the GPION clock */
rcu_periph_clock_enable(RCU_GPION);
rcu_periph_clock_enable(RCU_SYSCFG);
…
/* configure PN2 pin as input */
gpio_mode_set(GPION, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO_PIN_2);
…
/* enable and set pin EXTI interrupt to the lowest priority */
nvic_irq_enable(EXTI0_IRQn, 2U, 0U);
…
/* connect EXTI line to GPION2 pin */
syscfg_exti_line_config(EXTI_SOURCE_EXTI0_PN2);
…
/* configure EXTI line */
```

```
exti_init(EXTI_0, EXTI_INTERRUPT, EXTI_TRIG_FALLING);
exti_interrupt_flag_clear(EXTI_0);
```

2. PN2 can also be connected to EXTI24 to trigger an NMI interrupt. The configuration requires enabling PIN_NMIIE in SYSCFG first, and then configuring it as EXTI24 following the standard procedure.

Table 2-3. Example of of PN2 connect to EXTI24 configuration

```
/* enable the GPION clock */
rcu_periph_clock_enable(RCU_GPION);
rcu_periph_clock_enable(RCU_SYSCFG);
…
/* configure PN2 pin as input */
gpio_mode_set(GPION, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO_PIN_2);
…
/*enable PIN_NMIIE in SYSCFG first (required for EXTI24 NMI) */
syscfg_interrupt_enable(SYSCFG_INT_NMI_PIN);
…
/* configure EXTI24 line with falling edge trigger for NMI interrupt */
exti_init(EXTI_24, EXTI_INTERRUPT, EXTI_TRIG_FALLING);
exti_interrupt_flag_clear(EXTI_24);
```

2.5. General purpose timer (GPTIMERx)

2.5.1. Register write protection

To prevent miswriting registers, all registers have a write protection function, which is implemented by the GPTIMERx_WP register. The three registers GPTIMERx_SWEN, GPTIMERx_SWDIS and GPTIMERx_SWRST have independent bits SWEWPEN, SWDWPEN and SWRWPEN for write protection. All remaining registers are write-protected by WPEN bit.

To change whether write protection is enabled or not, the specified sequence "0xA5" is written to WPKEY[15:8] bits.

2.6. Cyclic redundancy checks management unit (CRC)

The cyclic redundancy check (CRC) code is an error-detecting code used in digital networks and storage devices to detect accidental changes to raw data. The CRC calculation unit can compute 7-, 8-, 16-, or 32-bit CRC checksums using a user-configurable polynomial.

In many applications, CRC-based techniques are also commonly used to verify the integrity of transmitted or stored data. According to the EN/IEC 60335-1 standard, these techniques provide a method for verifying the integrity of Flash memory. The CRC calculation unit helps compute the software signature during runtime and compare it with the reference signature generated at link time and stored in a designated memory location.

Note: CRC_FDATA is an independent 8-bit data register that does not participate in CRC computation.

The following example demonstrates the calculation process using the standard CRC-32 polynomial 0x04C11DB7:

- Polynomial (excluding the highest bit): poly = 0x04C11DB7
- Width: 32 bits
- Initial CRC: 0x00000000

For each incoming bit (MSB first), perform the following steps:

- top = (crc >> 31) & 1 // extract the most significant bit
- feedback = top ^ b // determine whether to “subtract” the polynomial
- crc = (crc << 1) & 0xFFFFFFFF // shift left by one bit
- if feedback == 1: crc ^= poly

If the input bit stream is 1, 0, 1, 1, the calculation proceeds as shown in the following table:

Table 2-4. The calculation process of CRC-32

Step	Input bit	Old CRC	MSB(top)	feedback = top⊕b	Left-shifted CRC	When feedback=1, XOR 0x04C11DB7	New CRC
initial	-	00000000	-	-	-	-	00000000
1	1	00000000	0	1	00000000	00000000 ⊕ 04C11DB7	04C11DB7
2	0	04C11DB7	0	0	09823B6E	-	09823B6E
3	1	09823B6E	0	1	130476DC	130476DC ⊕ 04C11DB7	17C56B6B
4	1	17C56B6B	0	1	2F8AD6D6	2F8AD6D6 ⊕ 04C11DB7	2B4BCB61

The final CRC result is: 0x2B4BCB61

2.7. Temperature sensor channel

2.7.1. Descriptions of peripheral configuration

A/D conversion of the temperature sensor channel can be selected for ADC2 only.

The GP1TEMP bit in ADC_CHSEL0 register selects A/D conversion of the temperature sensor channel in Group_pri1 scan once mode.

The GPxTEMP bit in ADC_CHSEL0/1 register selects A/D conversion of the temperature sensor channel of the according Group_pri x (Group_pri1/pri2/pri3/pri4) in groups scan mode, and the temperature sensor channel priority can be set in ADC_CHPRI0/1.

SPTTEMP[7:0] bits in ADC_SAMPR2 register are used to set the the sampling time of temperature sensor channel. If disconnection detection assist is used (CHRSEL[4:0] ≠ 5'b00000), sampling starts after discharging is completed during A/D conversion of the temperature sensor channel, and the discharging period CHRSEL[4:0] are fixed to 5'b11111 automatically. After the conversion is finished the CHRSEL[4:0] return to the setting before.

The temperature sensor can be used to measure the ambient temperature of the device. The sensor output voltage can be converted into a digital value by ADC. The sampling time for the temperature sensor is recommended to be set to at least t_{s_temp} μs (please refer to the datasheet.

To use the temperature sensor:

1. Configure the GPxTEMP bit in ADC_CHSEL0/1 register and the sampling time (t_{s_temp} μs or longer) for the channel by SPTTEMP[7:0] bits in ADC_SAMPR1 register.
2. Start the ADC conversion by the triggers.
3. Read the temperature data ($V_{temperature}$) in ADC_TEMPDATA register, and get the temperature with the following equation.

$$\text{Temperature (°C)} = \frac{(V_{temperature} - V_{-40}) * (-40-105)}{V_{-40} - V_{105}} - 40 \qquad (2\text{-}1)$$

V_{-40}: internal temperature sensor output voltage at -40°C, the typical value and factory calibration value address please refer to the datasheet (TS_CAL1).

V_{105}: internal temperature sensor output voltage at 105°C, the typical value and factory calibration value address please refer to the datasheet (TS_CAL1).

2.7.2. Example of temperature sensor usage

Table 2-5. Macro and variable definitions

```
/* address of the -40 degrees Celsius calibration value */
#define TEMPERATURE_TS_CAL_N40                    (0x1FFFF7F8)
/* address of the 105 degrees Celsius calibration value */
#define TEMPERATURE_TS_CAL2_105                   (0x1FFFF7FA)
```

```
/* temperature variable, unit: degrees Celsius */
__IO float temperature;
```

Table 2-6. RCU peripheral config

```
/* enable ADC clock */
rcu_periph_clock_enable(RCU_ADC2);
/* config ADC clock */
rcu_adc_clock_config(RCU_CK_ADCPRE_PCLK2, RCU_CK_ADCPRE_DIV6);
```

Table 2-7. ADC peripheral config

```
/* reset ADC */
adc_deinit(ADC2);
/* configure ADC data alignment */
adc_data_alignment_config(ADC2, ADC_DATAALIGN_RIGHT);
/* configure ADC resolution */
adc_resolution_config(ADC2, ADC_RESOLUTION_12B);
/* configure the ADC scan mode */
adc_group_scan_mode_config(ADC2, ADC_GROUP_PRI1_SCAN_ONCE);
/* deselect ADC group channel */
adc_group_channel_deselect(ADC2, ADC_GROUP_PRI1, ADC_CHANNEL_ALL);
/* select ADC channel */
adc_group_channel_config(ADC2, ADC_GROUP_PRI1, ADC_CHANNEL_TEMPERATURE, 0xFF);
/* enable ADC synchronous trigger */
adc_group_external_trigger_disable(ADC2, ADC_GROUP_PRI1);
/* enable ADC interface */
adc_enable(ADC2);
```

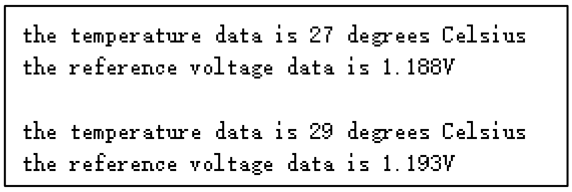
Table 2-8. Example of main function

```
/* calibration voltage at -40 degrees, unit: volts */
float V_n40 = 0;
/* calibration voltage at 105 degrees, unit: volts */
float V_105 = 0;
/* peripheral clocks configuration */
rcu_config();
/* configure systick */
systick_config();
/* ADC configuration */
adc_config();
/* UART configuration */
gd_eval_com_init(EVAL_COM);
/* obtain the chip calibration value. */
V_n40 = REG16(TEMPERATURE_TS_CAL_N40)* 5.0f / 4095;
V_105 = REG16(TEMPERATURE_TS_CAL2_105)* 5.0f / 4095;
while(1) {
    /* ADC software trigger enable */
```

```
        adc_group_software_trigger_enable(ADC2, ADC_GROUP_PRI1);
        /* delay a time in milliseconds */
        delay_1ms(2000);
        /* value convert */
        temperature = (adc_channel_data_read(ADC2, ADC_CHANNEL_TEMPERATURE) * 5.0f /
4095 - V_n40) * (-40-105) / (V_n40 - V_105) - 40;
        /* value print */
        printf(" the temperature data is %2.0f degrees Celsius\r\n", temperature);
        printf(" \r\n");
}
```

Connect the development board to the computer. Using a serial port assistant, you can receive the printed information from the temperature sensor routine, as shown in the *Figure 2-1. Serial port print information*:

Figure 2-1. Serial port print information



```
the temperature data is 27 degrees Celsius
the reference voltage data is 1.188V

the temperature data is 29 degrees Celsius
the reference voltage data is 1.193V
```

2.8. Port Output Controller (POC)

2.8.1. Precautions for the use of POC

Precautions for the POC module in the GD32M531 series are as follows:

(1) POC register bit attributes

Since the POC module is used to control the output of TIMER and GPTIMER channel pins, many register bits of the POC module have the rwo attribute for the sake of system reliability and safety. This means that after reset, these bits can only be written once. Therefore, during the initialization of the POC configuration, it is important to pay attention to the order of library function calls to prevent the accidental writing of rwo attribute register bits, which could prevent them from being modified a second time.

For example, the poc_input_dreq_status_config function should be called before the poc_input_polarity_config function and the poc_interrupt_enable / poc_interrupt_disable functions.

Table 2-9. Example of poc input configuration

```
/* configure POC_IN5 disabling request status */
poc_input_dreq_status_config(POC_IN5, POC_INn_DREQ_ENABLE);
…
/* configure POC_IN5 input polarity */
```

```
poc_input_polarity_config(POC_IN5, POC_INPUT_POLARITY_NONINVERTED);
…
/* enable/disable POC_IN5 input detection interrupt */
  poc_interrupt_enable(POC_INT_IN5);/ poc_interrupt_disable(POC_INT_IN5);
```

(2) POC input interrupt flag

When performing software configuration for POC pin input detection, first configure the POC detection-related functions, and then configure the POC pin for the corresponding GPIO AF function. Otherwise, the POC input interrupt flag may be set.

If the configuration is not done in the above order, you need to first configure the POC module and then configure the TIMER/GPTIMER module. Additionally, at the end of the POC module configuration, clear the corresponding input interrupt flag to prevent any impact on the output of the TIMER/GPTIMER channel pins.

In level detection mode, if you need to clear the corresponding POC input interrupt flag, the external pin must be at an invalid level; otherwise, it cannot be cleared.

2.9. GPTIMER Output Controller (GTOC)

2.9.1. Precautions for the use of GTOC

Precautions for the GTOC module in the GD32M531 series are as follows:

(1) GTOC register bit attributes

Since the GTOC module is used to control the output of GPTIMER channels, many register bits in the GTOC module have the rwo attribute for system reliability and safety considerations, which means they can only be written once after reset. Therefore, special attention should be paid to the calling sequence of library functions during GTOC initialization to prevent rwo register bits from being incorrectly written and thus unable to be modified again.

For example, the gtoc_output_closing_request_enable function should be placed before the gtoc_interrupt_flag_clear / gtoc_flag_clear functions.

Table 2-10. Example of GTOC configuration

```
/* enable GTOC0 output closing request */
gtoc_output_closing_request_enable(GTOC0, GTOC_OCR_SOURCE_GTOCPIN);
…
/* clear GTOC0 flag */
gtoc_interrupt_flag_clear(GTOC0,       GTOC_INT_FLAG_INIF);       /       gtoc_flag_clear(GTOC0,
GTOC_FLAG_INIF);
```

(2) GTOC input detection

In level input detection mode, when configured for active low, the corresponding GPIO's AF

(Alternate Function) must be configured, and the AF configuration must be placed before enabling the GTOC input detection . If not configured, requests will be continuously generated because the GTOCx_IN input remains low when the AF function is not configured.

When the external input pin of the GTOC is in low level input detection mode, the INIF flag will be set if the sampling frequency of the digital filter function is very low (fHCLK/512) after the configuration is completed. To avoid this situation, the recommended software configuration process for GTOC pin input detection is as follows:

1. Configure the GTOC pin for the corresponding GPIO AF function.
2. Configure the input polarity bit INPL in the GTOCx_CFG (x=0…3) register.
3. Configure the detection mode bit INDM in the GTOCx_CFG (x=0…3) register.
4. If the digital filter function is utilized, configure the DFSNUM[1:0] bits, DFSCDIV[3:0] bits, and the enable bit DFEN in the GTOCx_CFG (x=0…3) register.
5. Configure the detection enable bit INDEN in the GTOCx_CFG (x=0…3) register.

Additionally, in level detection mode, if you need to clear the interrupt flag INIF, the external pin must be at an invalid level; otherwise, it cannot be cleared.

2.10. Space vector pulse width modulation (SVPWM)

2.10.1. Precautions for the use of SVPWM

The input data of svpwm module is IEEE 32-Bit Single Precision Floating-Point Format. The SVPWM_UALPHA、SVPWM_UBETA registers are used to configure the input data.

The number of PWM cycles can be configured by setting the SVPWM_CAR register. And the output data of svpwm module are 16-bit unsigned number.

Table 2-11. Example of SVPWM data configuration

```
/* Define sin_data and cos_data as floating-point variables, and define hardware_ta, hardware_tb, and
hardware_tc as uint16_t variables. */
float sin_data, cos_data;
uint16_t hardware_ta, hardware_tb, hardware_tc;
```

Table 2-12. Example of SVPWM calculate

```
for(i = 0 ; i < 200; i++) {
    sin_data = 0.5 * sin((float)i / 200.0f * 2 * 3.1415926f);
    cos_data = 0.5 * cos((float)i / 200.0f * 2 * 3.1415926f);
    svpwm_alpha_beta_write(sin_data, cos_data);
    svpwm_enable();
    while(SET != svpwm_flag_get(SVPWM_FLAG_OSF));
    svpwm_ta_tb_tc_read(&hardware_ta, &hardware_tb, &hardware_tc);
    hareware_sector = (SECTOR)svpwm_sector_read();
}
```

2.11. Debug (DBG)

For specific content, refer to ***AN245 GD32M531xx GD-Link one line debugging description***.

3. Revision history

Table 3-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Mar.4, 2026

Important Notice