

GigaDevice Semiconductor Inc.

Device limitations of GD32H75E

Errata Sheet

Revision 1.0

(Oct. 2026)

Table of Contents

Table of Contents	2
List of Figures	4
List of Tables	5
1. Introduction	6
1.1. Revision identification.....	6
1.2. Summary of device limitations.....	6
2. Descriptions of device limitations.....	8
2.1. SYSTEM.....	8
2.1.1. When the system operates at high frequency and both the interrupt vector table and interrupt handling code are placed in ITCM, high-frequency interrupts may probabilistically cause false ITCM ECC error alarm	8
2.2. GPIO	8
2.2.1. After VDD and VDDA are powered off, if voltage continues to be injected into GPIO (except for PA9 / PA10 / PB12 / PB13), it may lead to an overcurrent risk in the chip's GPIO	8
2.3. TIMER	9
2.3.1. TIMER interrupt is at risk of triggering by mistake	9
2.4. USART	9
2.4.1. Data sample error occurs in LIN mode.....	9
2.4.2. In mute mode, the parity error caused by non-wake frames will set PERR bit.....	10
2.5. I2S.....	10
2.5.1. When I2S1 is used for communication and the GPIO and I2S1 configurations are in a special combination, it will probabilistically cause the I2S1_CK and I2S1_WS pins to continuously output signals.....	10
2.6. OSPI.....	11
2.6.1. When OSPI sends only data segments, the first clock is lost	11
2.6.2. When OSPI running clock is greater than 100MHz, read external memory status flag abnormal in status polling mode	11
2.7. EXMC.....	11
2.7.1. The bus may stuck during SDRAM access	11
2.7.2. Does not support unaligned address access	12
2.8. CAN	12
2.8.1. CAN RAM area may be tampered in receiving mailbox processing	12
2.8.2. After Bus off recovery, transmit error count is not cleared automatically	12
2.8.3. The mailbox data is not read in time may result in incorrect data reading for the current and next frames	13
	2

2.8.4.	When the CAN operating clock frequency is less than CK_APB2 and an error occurs in DLC segment, which will lead to receiving an incorrect frame ID	13
2.9.	ESC	14
2.9.1.	Under the default configuration, the external MCU (I2C master) cannot access the EEPROM via the EESDA / EESCL pins of the ESC chip	14
2.9.2.	IRQ interrupt handler fails to respond — interrupt cannot be triggered	14
2.9.3.	EtherCAT PHY link failure on simultaneous power-up	15
2.10.	Core	19
3.	Revision history	20

List of Figures

Figure 1-1. Device revision code of GD32H75E	6
--	---

List of Tables

Table 1-1. Applicable products	6
Table 1-2. Device limitations	6
Table 3-1. Revision history	20

1. Introduction

This document applies to GD32H75E product series, as shown in [Table 1-1. Applicable products](#). It offers technical guidance for using GD32MCU and provides workaround to current device limitations.

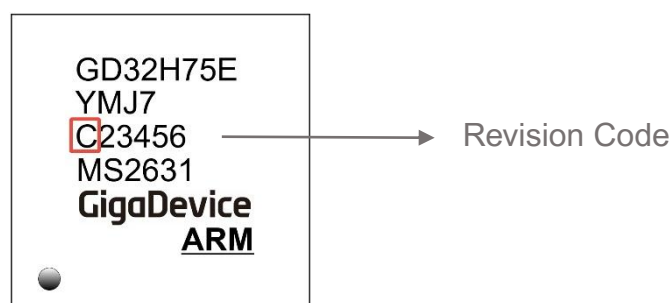
Table 1-1. Applicable products

Type	Part Numbers
MCU	GD32H75Exx series

1.1. Revision identification

The device revision can be identified according to the mark on the top of the package. The 1st code on Line 3 of the mark is the product revision code, as shown in [Figure 1-1. Device revision code of GD32H75E](#).

Figure 1-1. Device revision code of GD32H75E



1.2. Summary of device limitations

The device limitations of GD32H75E are shown in [Table 1-2. Device limitations](#), please refer to Section 2 for more details.

Table 1-2. Device limitations

Module	Limitations	Workaround
		Rev. Code C
SYSTEM	<i>When the system operates at high frequency and both the interrupt vector table and interrupt handling code are placed in ITCM, high-frequency interrupts may probabilistically cause false ITCM ECC error alarm</i>	Y
GPIO	<i>After VDD and VDDA are powered off, if voltage continues to be injected into GPIO (except for PA9 / PA10 / PB12 / PB13), it may lead to an overcurrent risk in the chip's GPIO</i>	Y
TIMER	<i>TIMER interrupt is at risk of triggering by mistake</i>	Y

Module	Limitations	Workaround
		Rev. Code C
USART	<i>Data sample error occurs in LIN mode</i>	Y
	<i>In mute mode, the parity error caused by non-wake frames will set PERR bit</i>	Y
I2S	<i>When I2S1 is used for communication and the GPIO and I2S1 configurations are in a special combination, it will probabilistically cause the I2S1_CK and I2S1_WS pins to continuously output signals</i>	Y
OSPI	<i>When OSPI sends only data segments, the first clock is lost</i>	Y
	<i>When OSPI running clock is greater than 100MHz, read external memory status flag abnormal in status polling mode</i>	Y
EXMC	<i>The bus may stuck during SDRAM access</i>	Y
	<i>Does not support unaligned address access</i>	Y
CAN	<i>CAN RAM area may be tampered in receiving mailbox processing</i>	Y
	<i>After Bus off recovery, transmit error count is not cleared automatically</i>	Y
	<i>The mailbox data is not read in time may result in incorrect data reading for the current and next frames</i>	Y
	<i>When the CAN operating clock frequency is less than CK_APB2 and an error occurs in DLC segment, which will lead to receiving an incorrect frame ID</i>	Y
ESC	<i>Under the default configuration, the external MCU (I2C master) cannot access the EEPROM via the EESDA / EESCL pins of the ESC chip</i>	Y
	<i>IRQ interrupt handler fails to respond — interrupt cannot be triggered</i>	Y
	<i>EtherCAT PHY link failure on simultaneous power-up</i>	Y

Note:

Y = Limitation present, workaround available

N = Limitation present, no workaround available

'--' = Limitation fixed

2. Descriptions of device limitations

2.1. SYSTEM

2.1.1. When the system operates at high frequency and both the interrupt vector table and interrupt handling code are placed in ITCM, high-frequency interrupts may probabilistically cause false ITCM ECC error alarm

Description & impact

When the system clock is configured to a high frequency (e.g., 600MHz), the AXI/AHB clock is configured to half the system clock frequency, and TCM inserts wait states (TCM_WAITSTATE = 1), placing both the interrupt vector table and interrupt handling code in ITCM may probabilistically cause false ITCM ECC error alarm when high-frequency interrupts occur.

Note: When the system clock exceeds 350MHz, TCM requires wait states to be inserted.

Workarounds

Use one of the following solutions:

- 1) Place the interrupt vector table in AXISRAM.
- 2) Avoid placing both the interrupt vector table and interrupt handling code in the same TCM block. For example, place the interrupt vector table in DTCM and the interrupt handling code in ITCM.
- 3) Adopt a frequency reduction scheme. Avoid simultaneous conditions where TCM_WAITSTATE = 1 and the AXI/AHB clock is configured to half the system clock frequency. For instance, configure both the system clock and AXI/AHB clock to 300MHz.

2.2. GPIO

2.2.1. After VDD and VDDA are powered off, if voltage continues to be injected into GPIO (except for PA9 / PA10 / PB12 / PB13), it may lead to an overcurrent risk in the chip's GPIO

Description & impact

When the chip's VDD and VDDA are powered off, if voltage (Vin) continues to be injected

externally into the GPIO (except for PA9 / PA10 / PB12 / PB13), due to the ESD protection circuit of the GPIO, the externally injected voltage may leak to VDD, forming a voltage of $V_{in}-0.7V$. If $V_{in}-0.7V$ is near the POR (1.6V), it may satisfy the POR conditions for the chip. However, due to the limited overcurrent capacity of the GPIO, this may lead to repeated POR/PDR events in the system, resulting in unpredictable risks for the system.

Workarounds

In system design, an undervoltage reset chip should be externally connected between the chip's VDD and NRST to ensure that when the voltage leaked from GPIO to VDD is below the threshold voltage of the undervoltage reset chip, it can directly pull down NRST to avoid erroneous operations. For a specific solution, you can refer to the "AN225 GD32H7xx Power Bypass Mode User Guide".

2.3. TIMER

2.3.1. TIMER interrupt is at risk of triggering by mistake

Description & impact

When the TIMER interrupt is enabled and the MCU operates at a high frequency, there is a risk that the corresponding TIMER interrupt flag may not be cleared in time due to the high code execution speed and the time required to clear the interrupt flag, leading to the risk of repeatedly entering the TIMER interrupt.

Workarounds

Clear the corresponding TIMER interrupt flag at the beginning of the TIMER interrupt handling function, and maintain at least 20 instruction cycles before exiting the interrupt.

2.4. USART

2.4.1. Data sample error occurs in LIN mode

Description & impact

When USART is in LIN mode as a receiver, data sample errors occur because the autobaud rate detection function is enabled before receiving the break frame.

Workarounds

- 1) Disable the autobaud rate detection function before receiving the break frame.
- 2) Enable the autobaud rate detection function after the LIN break detected flag (LBDF) is set.

2.4.2. In mute mode, the parity error caused by non-wake frames will set PERR bit

Description & impact

In mute mode, a parity error caused by a non-wake frame will result in a parity error (the PERR bit in the USART_STAT register is set) when no parity error is found in the wake frame.

Workarounds

The software ignores the parity error flag generated in this case.

2.5. I2S

2.5.1. When I2S1 is used for communication and the GPIO and I2S1 configurations are in a special combination, it will probabilistically cause the I2S1_CK and I2S1_WS pins to continuously output signals

Description & impact

I2S1_CK and I2S1_WS pins will probabilistically output signals continuously when I2S1 is used for communication and GPIO and I2S1 are configured in a special combination. The specific configuration and phenomenon are as follows:

GPIO configuration	I2S1 configuration	Phenomenon
GPIO_PUPD_PULLDOWN	I2S_CKPL_HIGH	I2S1_CK and I2S1_WS output signals continuously
GPIO_PUPD_NONE	I2S_CKPL_HIGH	I2S1_CK and I2S1_WS output signals continuously
GPIO_PUPD_PULLUP	I2S_CKPL_HIGH	I2S1 works normally
GPIO_PUPD_PULLDOWN	I2S_CKPL_LOW	I2S1 works normally
GPIO_PUPD_PULLUP	I2S_CKPL_LOW	I2S1 works normally
GPIO_PUPD_NONE	I2S_CKPL_LOW	I2S1 works normally

Workarounds

Please refer to the above for proper configuration of GPIO and I2S1.

2.6. OSPI

2.6.1. When OSPI sends only data segments, the first clock is lost

Description & impact

When OSPI sends only data segments, the first clock is lost, which results in the first data loss.

Workarounds

Send the first data as a command.

2.6.2. When OSPI running clock is greater than 100MHz, read external memory status flag abnormal in status polling mode

Description & impact

When the OSPI running clock is greater than 100MHz, the external memory may not be ready due to the hardware continuous polling interval is not enough, then OSPI will perform subsequent operations, which results in an exception.

Workarounds

Add a hardware delay (such as 20ms delay) before each poll of memory state.

2.7. EXMC

2.7.1. The bus may stuck during SDRAM access

Description & impact

When accessing SDRAM, if the SDRAM data bit width is 32 bits, the bus sends two 8-bit or two 16-bit accesses, and then sends a 64-bit access, if the 64-bit access data happens to be composed of the previous two 8-bit or 16-bit accesses, the bus may stuck.

Workarounds

Use one of the following solutions:

- 1) Disable burst access to SDRAM (clear BRSTRD bit of EXMC_SDCTL0/1 register)
- 2) Enable the CPU cache function

2.7.2. Does not support unaligned address access

Description & impact

The bus does not support unaligned address access to SDRAM.

Workarounds

Enable the cache function of the CPU before accessing SDRAM.

2.8. CAN

2.8.1. CAN RAM area may be tampered in receiving mailbox processing

Description & impact

If the global mailbox unlocking operation is not performed in the receiving mailbox processing routine (due to incorrect operation in the software), there is a certain probability that the CAN RAM area will be tampered, which will cause data transmission and reception exceptions.

Workarounds

Wait for the sending completion flag in CAN_STAT register to be set instead of judging by the CODE segment value of the sent mailbox before every data transmission. The reference code is as follow:

```
Flagstatus can_tx_state = RESET;
{
    if((RESET == can_tx_state) || (SET == can_flag_get(CAN1, CAN_FLAG_MB1))){
        can_tx_state = SET;
        can_flag_clear(CAN1, CAN_FLAG_MB1);
        /* transmit message */
        can_mailbox_config(CAN1, 1, &transmit_message);
        /* user code */
    }
}
```

2.8.2. After Bus off recovery, transmit error count is not cleared automatically

Description & impact

After Bus off recovery, transmit error count (TECNT) is not cleared automatically by hardware.

Workarounds

After Bus off recovery, using software method to clear the transmit error count (TECNT). Taking CAN1 as an example, the reference code is as follow.

```
{
```

```

/* bus off recovery flag is set */
if(RESET != can_flag_get(CAN1, CAN_FLAG_BUSOFF_RECOVERY))
{
    /* enter inactive mode */
    can_operation_mode_enter(CAN1, CAN_INACTIVE_MODE);
    /* clear transmit error count */
    CAN_ERR0(CAN1) &= ~(CAN_ERR0_TECNT);
    /* enter normal mode */
    can_operation_mode_enter(CAN1, CAN_NORMAL_MODE);
    can_flag_clear(CAN1, CAN_FLAG_BUSOFF_RECOVERY);
}
}

```

2.8.3. The mailbox data is not read in time may result in incorrect data reading for the current and next frames

Description & impact

When handling mailbox receive, if a new CAN frame (the next frame) is moved into the receive mailbox while reading this receive mailbox data (the current frame), it may lead to incorrect reading of the current frame and the next frame data.

Note: Only the current frame and the next frame are affected.

Workarounds

Use one of the following solutions:

- 1) Use mailbox reception interrupt and configure it to highest priority. When a receive mailbox interrupt occurs, promptly read and process the mailbox data (before the end of the next frame).
- 2) Use CAN FIFO reception instead of mailbox reception.
- 3) Enable the mailbox queue by setting the RPFQEN bit and set the mailbox reception interrupt to the highest priority.

2.8.4. When the CAN operating clock frequency is less than CK_APB2 and an error occurs in DLC segment, which will lead to receiving an incorrect frame ID

Description & impact

When the CAN operating clock frequency is less than CK_APB2, if external interference on the CAN bus causes an error frame to be sent due to an error detected in the DLC segment while receiving a frame, this will result in the subsequent frame receiving the error frame ID

(even if this error frame ID is filtered by the CAN filter).

Workarounds

Use one of the following solutions:

- 1) The software configures CAN clock source as CK_APB2. Taking CAN1 as an example, the reference code is as follow:

```
{  
    /* configure the CAN1 clock source as CK_APB2 */  
    rcu_can_clock_config(IDX_CAN1, RCU_CANSRC_APB2);  
}
```

- 2) Software checks the frame ID (applicable when the CAN operating clock frequency is less than CK_APB2). When the CAN mailbox receives a non-target frame ID and an error occurs, the software needs to reconfigure the CAN reception mailbox parameters. The specific configuration steps are as follows:
 - a) Enter inactive mode
 - b) Reconfigure the reception mailbox parameters
 - c) Enter normal mode

2.9. ESC

2.9.1. Under the default configuration, the external MCU (I2C master) cannot access the EEPROM via the EESDA / EESCL pins of the ESC chip

Description & impact

When using an external MCU to access the EEPROM, the EESDA / EESCL pins of the ESC chip are set to push-pull mode by default, which prevents the external MCU from reading EEPROM data. In this case, the ESC chip can still access the EEPROM normally, and its functionality remains unaffected.

Workarounds

The external MCU can access the EEPROM only after configuring the EESDA / EESCL pins of the ESC chip to open-drain output mode through the PDI interface.

2.9.2. IRQ interrupt handler fails to respond — interrupt cannot be triggered

Description & impact

When the ESC module of the GD32H75E series MCU operates in SM mode, if the master exhibits large jitter in its frame transmission period and the interval between two adjacent frames is shorter than the ESC interrupt processing time, the IRQ interrupt can no longer be triggered. As a result, the slave can no longer refresh the PDO data sent by the master.

Switching back to the OP state restores data refreshing.

Workarounds

PJ10 is the pin that connects the ESC hardware IRQ to the MCU. Add a PJ10 pin level check inside the IRQ interrupt service routine: after executing `PDI_Isr()` and clearing the ESC interrupt flag, if PJ10 is still at a low level, it indicates that there are still unhandled interrupts. In that case, call `PDI_Isr()` again to process the pending events, and clear the interrupt flag once more.

Reference code is as follows:

```
/* code example */
void EcatIsr(void)
{
    if(exti_interrupt_flag_get(EXTI_10) != RESET) {

        ACK_ESC_INT;
        PDI_Isr();

        if(gpio_input_bit_get(GPIOJ, GPIO_PIN_10) == RESET) {
            PDI_Isr();
            ACK_ESC_INT;
        }
    }
}
```

2.9.3. EtherCAT PHY link failure on simultaneous power-up

Description & impact

When using the built-in ESC module of the GD32H75E series MCU, if two (or more) nodes on the bus have exactly the same power-up timing, the FLP auto-negotiation pulses of the built-in PHY are generated from pseudo-random timing, so after power-on reset these nodes share exactly the same pseudo-random initial phase — the upstream and downstream PHYs transmit FLPs at the same time, causing pulse collisions and preventing auto-negotiation from completing link-up immediately.

The two nodes can only gradually separate their phases through crystal oscillator frequency deviation; the link can be established only after their FLP transmission timings drift apart. This manifests as slow Link-Up on some slaves.

Workarounds

After the software starts up on power-on, generate a true random delay in the range of 0–65535 μ s. Once the delay expires, perform a PHY software reset and then start the EtherCAT protocol stack.

By artificially offsetting each node's PHY reset starting point, the FLP pulse transmission phases are dispersed, avoiding synchronized collisions. This allows the PHYs to complete auto-negotiation quickly and establish a stable link even when multiple nodes power up

simultaneously.

Reference code is as follows:

```

/*!
 \brief      enable esc func
 \param[in]  none
 \param[out] none
 \retval     none
 */
static void esc_enable(void)
{
    volatile uint32_t temp = 0;

    ospi_hw_init(OSPI_INTERFACE, &ospi_init_struct);
    ospi_enable_ospi_mode(OSPI_INTERFACE, &ospi_init_struct, SPI_MODE);

    /*ByteTest and PMU ready*/
    ospi_read(PMU_PDIREFVAL, (uint8_t*)&temp, 4);
    if(temp != 0x76543210)
    {
        while(1);
    }

    trng_val= trng_get();
    delay_lms(20); /* ensure the PHY hardware reset time */
    delay_us(trng_val); /* 0-65535us random delay to stagger the PHY reset among slaves */

    temp = 0x307;
    ospi_write(ESC_RCU_RSTCFG, (uint8_t *)&temp, 4);
    delay_lms(5);

    ospi_enable_ospi_mode(OSPI_INTERFACE, &ospi_init_struct, SPI_MODE);

    ospi_read(PMU_CTRL0, (uint8_t*)&temp, 4);
    if( (temp & BIT(0)) != BIT(0))
    {
        while(1);
    }
    esc_oeext_clock_disable();
}

```

```

/*!
 \brief      check whether the TRNG module is ready
 \param[in]  none
 \param[out] none
 \retval     ErrStatus: SUCCESS or ERROR
 */
ErrStatus trng_ready_check(void)
{
    uint32_t timeout = 0;
    FlagStatus trng_flag = RESET;

    /* check whether the random data is valid */
    do {
        timeout++;
        trng_flag = trng_flag_get(TRNG_FLAG_DRDY);
    } while((RESET == trng_flag) && (0xFFFFU > timeout));

    if((SET == trng_flag)
        && (RESET == trng_interrupt_flag_get(TRNG_INT_FLAG_CEIF))
        && (RESET == trng_interrupt_flag_get(TRNG_INT_FLAG_SEIF))
        && (RESET == trng_flag_get(TRNG_FLAG_CECS))
        && (RESET == trng_flag_get(TRNG_FLAG_SECS))
        && (OU == (TRNG_STAT & TRNG_STAT_ERRSTA))) {
        return SUCCESS;
    }

    /* return check status */
    return ERROR;
}

```

```

/*!
 \brief      configure the TRNG module parameters
 \param[in]  none
 \param[out] none
 \retval     ErrStatus: SUCCESS or ERROR
 */
ErrStatus trng_configuration(void)
{
    uint32_t timeout = 0;

    /* configure the internal 48M high speed clock for TRNG */
    rcu_osc_on(RCU_IRC48M);
    /* wait until IRC48M clock is stable */
    while((RESET == rcu_flag_get(RCU_FLAG_IRC48MSTB)) && (0xFFFFU > timeout)) {
        timeout++;
    }
    if(RESET == rcu_flag_get(RCU_FLAG_IRC48MSTB)) {
        return ERROR;
    }

    /* select the internal IRC48M as the 48M system clock source */
    rcu_ck48m_clock_config(RCU_CK48MSRC_IRC48M);

    /* enable the TRNG peripheral clock */
    rcu_periph_clock_enable(RCU_TRNG);
    /* enable the TRNG clock error detection */
    trng_clockerror_detection_enable();

    /* reset the TRNG registers to default configuration */
    trng_deinit();
    /* enable the post-processing conditioning module reset */
    trng_conditioning_reset_enable();
    /* configure the TRNG working mode as NIST security mode */
    trng_mode_config(TRNG_MODSEL_NIST);
    /* configure the conditioning input bit width: 440bit raw entropy data */
    trng_conditioning_input_bitwidth(TRNG_INMOD_440BIT);
    /* configure the conditioning output bit width: 256bit, output eight 32-bit random numbers */
    trng_conditioning_output_bitwidth(TRNG_OUTMOD_256BIT);
    /* configure the conditioning crypto algorithm as SHA256 hash algorithm */
    trng_conditioning_algo_config(TRNG_ALGO_SHA256);
    /* enable the TRNG post-processing conditioning module */
    trng_conditioning_enable();
    /* enable the TRNG security post-processing */
    trng_postprocessing_enable();

    /* enable the TRNG hardware module */
    trng_enable();

    return SUCCESS;
}

```

```

/*!
\brief      get a 16-bit true random number from the TRNG module
\param[in]   none
\param[out]  none
\retval     low 16 bits of the random data; SysTick-derived value on failure
\note       The CK48M clock source and the IRC48M enable state are backed up
            before the TRNG is configured and restored before returning, so
            the system clock and other clock domains (e.g. USB) are left
            unchanged after this function returns.
*/
uint16_t trng_get(void)
{
    uint8_t retry = 0;
    uint16_t result = 0U;
    ErrStatus random_status = ERROR;
    /* backup the current CK48M clock source and IRC48M enable state */
    uint32_t ck48m_src_bak = RCU_ADDCTL0 & RCU_ADDCTL0_CK48MSEL;
    uint32_t irc48m_en_bak = RCU_ADDCTL0 & RCU_ADDCTL0_IRC48MEN;
    uint32_t trng_clk_en_bak = RCU_AHB2EN & RCU_AHB2EN_TRNGEN;

    /* retry random data acquisition up to three times */
    while((retry < 3U) && (ERROR == random_status)) {
        retry++;
        if(SUCCESS == trng_configuration()) {
            if(SUCCESS == trng_ready_check()) {
                /* read the valid data before disabling the TRNG */
                random_data = trng_get_true_random_data();
                result = (uint16_t)(random_data & 0xFFFF); /* low 16 bits of the random data */
                random_status = SUCCESS;
            }

            trng_disable();
        }
    }

    if(ERROR == random_status) {
        /* avoid identical fixed delays when TRNG acquisition fails */
        result = (uint16_t)(SysTick->VAL & 0xFFFFU);
    }

    /* restore the TRNG peripheral clock enable state */
    if(0U == trng_clk_en_bak) {
        rcu_periph_clock_disable(RCU_TRNG);
    }
    /* restore the CK48M clock source to its original setting */
    rcu_ck48m_clock_config(ck48m_src_bak);
    /* turn IRC48M off again if it was originally disabled */
    if(0U == irc48m_en_bak) {
        rcu_osc_off(RCU_IRC48M);
    }

    return result;
}

```

2.10. Core

About Cortex-M7 limitations, please refer to "Cortex-M7 (AT610) and Cortex-M7 with FPU (AT611) Software Developer Errata Notice". This document can be downloaded on ARM official website.

3. Revision history

Table 3-1. Revision history

Revision No.	Description	Date
1.0	Initial Release	Oct.27 2026

Important Notice

This document is the property of GigaDevice Semiconductor Inc. and its subsidiaries (the "Company"). This document, including any product of the Company described in this document (the "Product"), is owned by the Company according to the laws of the People's Republic of China and other applicable laws. The Company reserves all rights under such laws and no Intellectual Property Rights are transferred (either wholly or partially) or licensed by the Company (either expressly or impliedly) herein. The names and brands of third party referred thereto (if any) are the property of their respective owner and referred to for identification purposes only.

To the maximum extent permitted by applicable law, the Company makes no representations or warranties of any kind, express or implied, with regard to the merchantability and the fitness for a particular purpose of the Product, nor does the Company assume any liability arising out of the application or use of any Product. Any information provided in this document is provided only for reference purposes. It is the sole responsibility of the user of this document to determine whether the Product is suitable and fit for its applications and products planned, and properly design, program, and test the functionality and safety of its applications and products planned using the Product. The Product is designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only, and the Product is not designed or intended for use in (i) safety critical applications such as weapons systems, nuclear facilities, atomic energy controller, combustion controller, aeronautic or aerospace applications, traffic signal instruments, pollution control or hazardous substance management; (ii) life-support systems, other medical equipment or systems (including life support equipment and surgical implants); (iii) automotive applications or environments, including but not limited to applications for active and passive safety of automobiles (regardless of front market or aftermarket), for example, EPS, braking, ADAS (camera/fusion), EMS, TCU, BMS, BSG, TPMS, Airbag, Suspension, DMS, ICMS, Domain, ESC, DCDC, e-clutch, advanced-lighting, etc.. Automobile herein means a vehicle propelled by a self-contained motor, engine or the like, such as, without limitation, cars, trucks, motorcycles, electric cars, and other transportation devices; and/or (iv) other uses where the failure of the device or the Product can reasonably be expected to result in personal injury, death, or severe property or environmental damage (collectively "Unintended Uses"). Customers shall take any and all actions to ensure the Product meets the applicable laws and regulations. The Company is not liable for, in whole or in part, and customers shall hereby release the Company as well as its suppliers and/or distributors from, any claim, damage, or other liability arising from or related to all Unintended Uses of the Product. Customers shall indemnify and hold the Company, and its officers, employees, subsidiaries, affiliates as well as its suppliers and/or distributors harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of the Product.

Information in this document is provided solely in connection with the Product. The Company reserves the right to make changes, corrections, modifications or improvements to this document and the Product described herein at any time without notice. The Company shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. Information in this document supersedes and replaces information previously supplied in any prior versions of this document.