Application Note

Document No.: AN1170

G32A1085 A1065 A1045

FLASH User Guide

Version: V1.0

1 Introduction

This application note explains how to use FLASH on the G32A1085 A1065 A1045 series. It covers FLASH ECC verification, the FLASH operation workflow, and the procedures for common FLASH functions.

Content

1 Introduction 1

2 Chip FLASH Information 3

3 ECC Functions 4

3.1 FLASH ECC Function 4

3.2 SRAM ECC Function 4

3.3 CAN SRAM ECC Function 4

3.4 ECC Notes 4

4 FLASH Unlock 6

4.1 FLASH Unlock 6

4.2 Option Byte Unlock 7

5 FLASH Erase, Write, and Read 8

5.1 FLASH Erase 8

5.2 FLASH Programming 10

5.3 Key SDK Function Descriptions 12

6 FLASH Protection 13

6.1 FLASH Read/Write Protection 13

6.2 Option Byte Programming 14

7 FLASH Precautions 16

7.1 Recommendation 1 16

7.2 Recommendation 2 17

7.3 Recommendation 3 17

8 Revision History 18

2 Chip FLASH Information

The table below lists the FLASH and memory sizes for each chip in the G32A1085 A1065 A1045 series:

Table 1 Memory Description

Memory	G32A1085 Max Capacity	G32A1065 Max Capacity	G32A1045 Max Capacity	Description
PFlash	256KB	128KB	64KB	Stores user code and constant data
DFlash	32KB	32KB	16KB	Data storage area
SRAM	32KB	16KB	8KB	—
Option Bytes	24Bytes	24Bytes	24Bytes	Configures main memory read/write protection and MCU operating mode

3 ECC Functions

3.1 FLASH ECC Function

Both PFLASH and DFLASH support ECC. The FLASH uses a single-error-correcting, double-error-detecting SECDED ECC algorithm to verify data integrity and improve reliability. ECC is enabled by default after a reset. It can be disabled by clearing the ECCEN bit.

When ECC is enabled, and an uncorrectable ECC error is detected, the DBFIFLG error flag is set. If DBFIEN is enabled, an interrupt is generated.

After an uncorrectable ECC error occurs, the address of the latest error is stored in the FLASH_ECC_ADDR register. When the DBFIFLG flag is cleared, the recorded error address is cleared as well.

3.2 SRAM ECC Function

The SMS provides ECC verification for SRAM. It supports error detection, single-bit error correction, and double-bit error detection. Extra parity bits are added for each address. When ECC is enabled through ECC_EN, the SMS calculates parity bits internally for every write operation.

For every read operation, including read-modify-write operations, ECC errors are reported and recorded. The address of the latest error is stored in the SMS_ECC_LOGn register. After reset, the interrupt mask bits in SMS_INTRMn remain set to 1. Clearing the corresponding mask bit allows reported ECC errors to generate a maskable system interrupt.

After INSERT_ERR is configured, the SMS can inject error bits during SRAM accesses for diagnostics. Single-bit, double-bit, and triple-bit error injection are supported.

3.3 CAN SRAM ECC Function

CAN SRAM also supports ECC verification, including single-bit error correction and double-bit error detection. A verification error can trigger an interrupt, and the error address can be read.

After ECC_EN is enabled for CAN SRAM ECC checking and the global ECC interrupt is enabled, ECC errors are detected and recorded on every CAN SRAM access. The BEC_FLAG or BEU_FLAG error flag is set, and the address of the latest error is stored in the ERR_ADDR register. If BEC_INT_EN or BEU_INT_EN is set, the corresponding ECC error generates an interrupt.

3.4 ECC Notes

After data verification is enabled, the ECC controller generates a 7-bit ECC code for each 32-bit data write. During each read, all 39 bits, including 32 data bits and 7 ECC bits, are verified. ECC verification is not performed during writes. If a verification error is detected, the related status flags are set, and an interrupt request is generated.

(1) Enable the ECC error interrupt. In the interrupt handler, erase the FLASH address where

the error occurred to restore access to that address.

(2) After an ECC error, erase the affected data. To prevent data loss, use a second FLASH sector as a backup. When an ECC error occurs, read the data from the backup sector. In practice, after erasing or writing one sector, also erase or write the backup sector. The backup sector should only be read at power-on or when an ECC error occurs. Make sure to identify which FLASH sector caused the ECC error and erase only that sector. A marker can be written before reading the address to help identify the source sector.

(3) A hidden erase command may be added to the firmware and triggered through a hidden serial command or another communication interface. This can help protect the chip from malicious code readout and can also allow the chip to be recovered and its security state cleared when a firmware update is required.

4 FLASH Unlock

4.1 FLASH Unlock

After reset, the FLASH Memory Controller, or FMC, is hardware-locked and cannot be written directly. To unlock the FMC, write the correct key sequence to FMC_KEY in the required order:

- KEY1=0x4567 0123
- KEY2=0xCDEF 89AB

If the keys are written in the wrong order or with incorrect values, a hardware fault occurs, and the FMC remains locked. All FMC operations are invalid until the next reset. The FMC can also be locked by software by setting the LOCK bit in Control Register 2 (FMC_CTRL2) to 1.

For each FLASH programming operation, use the following sequence: FLASH Unlock → User Programming → FLASH Lock. This prevents accidental modification of user code or data after the programming operation is complete.

In AUTOSAR Fls, FLASH unlock is handled inside the low-level LLD hardware operation functions. FLASH is unlocked only when needed, such as during erase, write, or option byte modification, and is locked again immediately after the operation completes. This prevents FLASH from being left unprotected. The main function only starts and schedules tasks, so no separate unlock step is required when using the SDK.

AUTOSAR Fls unlock call chain: Main function starts task → Fls_MainFunction() schedules task → corresponding LLD function runs → unlock + hardware operation + lock are executed inside the function.

AUTOSAR Fls Unlock/Lock Execution Flow:

Main: Fls_Erase → Fls_MainFunction() → calls Fls_SectorErase()

↓

[Unlock executed here] Inside Fls_SectorErase():

Write 0x45670123, then 0xCDEF89AB to FMC->KEYR

REG_WRITE32(FMC_KEY_REG_ADDR, FMC_KEY_1);

REG_WRITE32(FMC_KEY_REG_ADDR, FMC_KEY_2); → FLASH Unlocked

↓

Select sector → Start erase → Wait for completion by polling busy flag → Check errors

↓

REG_BIT_SET32(FMC_CTRL2_REG_ADDR, FMC_CTRL2_LOCK_MASK); → Immediately re-locked

4.2 Option Byte Unlock

By default, option bytes are always readable but write-protected. To program or erase the option byte block, write the correct key sequence to FMC_OBKEY first. The keys are the same as the FLASH unlock keys. This enables write access to the option byte block. The OBWEN bit in FMC_CTRL2 indicates whether write access is enabled. Clearing OBWEN disables write access.

After a system reset, option bytes are locked by default. They can only be modified after a correct unlock sequence. The difference from FLASH unlock is that the keys are written to FMC_OBKEY instead of FMC_KEY.

- KEY1=0x4567 0123
- KEY2=0xCDEF 89AB

After each option byte modification, a forced reload by setting OBLOAD or a power cycle is required before the changes take effect.

In AUTOSAR Fls, option byte unlocking is handled by the lower-level LLD hardware functions. The option bytes are unlocked only when needed, such as during erase, write, or option byte modification, and are locked again immediately after the operation. This prevents them from being left unprotected. The main function only starts and triggers tasks, so no separate unlock step is required when using the SDK.

5 FLASH Erase, Write, and Read

5.1 FLASH Erase

The G32A1085 A1065 A1045 series supports page erase and mass erase. Mass erase can be selected for PFLASH, DFLASH, or both. An erase operation must be performed before each write. If the target address is not erased to all 1s, a programming error is triggered.

Page Erase Procedure:

(1) Unlock FLASH and check the BUSYF flag to confirm that no FLASH operation is in progress.

(2) Set the PAGEERA bit.

(3) Write any address within the target erase page to FMC_ADDR.

(4) Set the STA bit.

(5) Wait until BUSYF is cleared, then perform read-back verification on the erased address.

Mass Erase Procedure:

Mass erase can erase all contents of PFLASH, DFLASH, or both. Use this operation carefully to avoid unintended data loss.

(1) Unlock FLASH and check the BUSYF flag to confirm that no FLASH operation is in progress.

(2) Set the MASSERA bit.

(3) Configure the MER_TYPE bit to select the erase region.

(4) Set the STA bit.

(5) Wait until BUSYF is cleared, then perform read-back verification on the erased address.

Notes:

Single-page FLASH erase takes about 4ms. Multi-page erase time is 4ms × number of pages. Mass erase takes about 10ms per FLASH type. Because PFLASH and DFLASH use independent interfaces, erasing both together takes about 20ms in total. The minimum programming unit is one double-word, and programming takes about 215μs.

Three important cautions:

(1) Power-On Delay: After the chip powers on, wait 100ms for the system to stabilize before accessing FLASH.

(2) Watchdog Handling: During consecutive multi-page erase operations, feed the watchdog or clear the watchdog counter throughout the erase process. This prevents a watchdog reset

caused by the long erase time.

(3) Power Supply Stability: The power supply must remain stable during erase and write operations. Voltage monitoring is recommended. If the supply voltage is unstable or abnormal, stop all FLASH operations immediately. Otherwise, FLASH may malfunction and later trigger ECC errors when the affected address is accessed after the next power-on.

Before calling Fls_SectorErase(), call Fls_GetStatus() to confirm that FLASH is idle. If FLASH is not idle, the erase function returns ERROR immediately.

Application Code Example:

```
VAR( Std_ReturnType, AUTOMATIC ) u8RetVal = (Std_ReturnType)E_NOT_OK;
VAR( MemIf_StatusType, AUTOMATIC ) eRetVal = MEMIF_IDLE;
VAR( MemIf_JobResultType, AUTOMATIC ) eJobRetVal = MEMIF_JOB_FAILED;
/* Set the erase operation, erase 1KB Data Flash starting from address 0x0803E000 */
u8RetVal = Fls_Erase(0x3E000, 1024);
eJobRetVal = Fls_GetJobResult();
eRetVal = Fls_GetStatus();
if((E_OK == u8RetVal) && (MEMIF_JOB_PENDING == eJobRetVal) && (MEMIF_BUSY == eRetVal))
{
    /* Erase 1KB Data Flash starting from address 0x0803E000 */
    while (MEMIF_IDLE != Fls_GetStatus())
    {
        Fls_MainFunction();
    }
}
```

AUTOSAR Fls Erase Execution Flow:

Main: Fls_Erase(0x3E000, 1024) → Submit erase request → Save task parameters

→ Set s_jobResult = MEMIF_JOB_PENDING → Set module state to MEMIF_BUSY

↓

Main: Enter loop while module state is BUSY

↓

Loop: Repeatedly call Fls_MainFunction() → Detect pending erase task

→ Call LLD Fls_SectorErase() to perform hardware erase

↓

Hardware erase complete → Fls_MainFunction() sets s_jobResult = MEMIF_JOB_OK

→ Set module state to MEMIF_IDLE

↓

Loop condition is no longer true, because MEMIF_IDLE == Fls_GetStatus() → Exit loop → Done

5.2 FLASH Programming

For the G32A1085 A1065 A1045 series, FLASH write addresses must be 64-bit aligned. Otherwise, the PAEF flag is set. The programming length is configured through PROGLEN. The minimum programming length is 64 bits, or 8 bytes.

Before writing, confirm that the destination address has already been erased. If the destination address has not been erased, the write is invalid, and the PEF bit in FMC_STS is set. If the destination address is write-protected, the write is invalid, and a write protection error occurs. In this case, the WPEF bit in FMC_STS is set.

Programming Procedure:

(1) Unlock FLASH and check the BUSYF flag to confirm that no FLASH operation is in progress.

(2) Set the PG bit.

(3) Write the programming start address to FMC_ADDR. The address must be double-word aligned.

(4) Configure the programming length with PROGLEN. Then write the data sequentially to FLASH_PROG_DATAx. For example, to write 0x12345678_22345678, first write 0x22345678 to FLASH_PROG_DATA0, then write 0x12345678 to FLASH_PROG_DATA1. For more data, continue writing to the following FLASH_PROG_DATA registers.

(5) Set the STA bit.

(6) Wait until BUSYF is cleared, then perform read-back verification on the written address.

Notes:

Because page erase clears all data in the page, any valid data in the same erase page must be read into a buffer before the erase and written back after the erase completes. Make sure FLASH is idle before writing. If FLASH is not idle, the write function returns ERROR immediately.

Application Code Example:

```
uint32_t dataNum = 0;
uint8 g_aaAppBufferForWriteB[1024];
for(dataNum = 0;dataNum < 1024;dataNum++)
{
    g_aaAppBufferForWriteB[dataNum] = dataNum;
}
/* Set the write operation, write 1KB Data Flash starting from address 0x0803E000 */
u8RetVal = Fls_Write(0x3E000, (const uint8 *)&g_aaAppBufferForWriteB[0], 1024);
if(E_OK == u8RetVal)
{
    while (MEMIF_IDLE != Fls_GetStatus())
    {
        Fls_MainFunction();
    }
}
```

AUTOSAR Fls Programming Execution Flow:

Main: Fls_Write(0x3E000, &g_aaAppBufferForWriteB[0], 1024) → Submit write request

→ Save task parameters → Set s_jobResult = MEMIF_JOB_PENDING → Set module state to MEMIF_BUSY

↓

Main: Enter loop while module state is BUSY

↓

Loop: Repeatedly call Fls_MainFunction() → Detect pending write task

→ Call LLD Fls_SectorWrite() to perform double-word-aligned programming

↓

Hardware write complete → Fls_MainFunction() sets s_jobResult = MEMIF_JOB_OK

→ Set module state to MEMIF_IDLE

↓

Loop condition is no longer true, because MEMIF_IDLE == Fls_GetStatus() → Exit loop → Done

5.3 Key SDK Function Descriptions

Fls_MainFunction(): The background main function of the AUTOSAR Fls module. It internally performs all FLASH hardware operations. The execution flow is as follows:

(1) Check whether the global task status s_jobResult is MEMIF_JOB_PENDING.

(2) If a task is pending, call the corresponding LLD hardware function based on s_jobOpnType, which indicates the task type: erase, write, read, compare, or blank check:

- Erase: Call Fls_SectorErase(); unlock FLASH → sector erase → lock FLASH.

- Write: Call Fls_SectorWrite(); unlock FLASH → program double-word by double-word → lock FLASH.

- Read: Call Fls_SectorRead(); direct memory-mapped read, no unlock required.

- Compare: Call Fls_SectorCompare(); read FLASH and compare byte by byte.

(3) Blank Check: Call Fls_SectorBlankCheck(); read FLASH and verify that all bytes are 0xFF. After the hardware operation completes, update “s_jobResult” based on the LLD return value.

(4) Set the Fls module state to “MEMIF_IDLE” to indicate that the task is complete.

(5) Depending on the task result, call the user-configured job completion notification or job error notification callback.

6 FLASH Protection

6.1 FLASH Read/Write Protection

FLASH read and write protection can be configured through option bytes.

Read Protection

Read protection is configured through the READPROT option byte. It supports three levels: Level 0, Level 1, and Level 2, as described below:

Table 2 Read Protection Level Differences

Level	READPROT	Description
Level 0	0xAA	Main memory and option bytes can be erased, written, and read.
Level 1	Any value other than 0xAA or 0xCC	User mode: main memory and option bytes can be erased, written, and read. Debug/SRAM run mode: access to main memory is prohibited, except for mass erase. Option bytes can be erased, written, and read. When downgrading to Level 0, main memory is mass erased first.
Level 2	0xCC	Debug mode and SRAM run mode are disabled. Main memory can be erased, written, and read. Option bytes can be read. Option bytes outside the read protection level can be written. Erasing option bytes is not allowed.

Write Protection

Write protection for individual pages in main memory is configured through the PWRP and DWRP option bytes. After write protection is enabled, the protected pages cannot be modified by any method. For PFLASH, the basic write protection unit is 16 pages, or 8KB. For DFLASH, the basic write protection unit is 2 pages, or 1KB.

Table 3 Main Memory Write Protection WRPx Function Description

Product	Function Description
G32A1085 A1065 A1045	Each bit in WRPx controls write protection for an 8 KB, 16-page, or 1 KB, 2-page region of main memory. 0: Write protection enabled. 1: Write protection disabled. WRP0: PFLASH write protection for pages 0–127. WRP1: PFLASH write protection for pages 128–255, G32A1085 and G32A1065 only. WRP2: PFLASH write protection for pages 256–383, G32A1085 only. WRP3: PFLASH write protection for pages 384–511, G32A1085 only. WRP4: DFLASH write protection for pages 0–15. WRP5: DFLASH write protection for pages 16–31. WRP6: DFLASH write protection for pages 32–47, G32A1085 and G32A1065 only. WRP7: DFLASH write protection for pages 48–63, G32A1085 and G32A1065 only.

Note: FLASH read protection and write protection are configured independently. Removing write protection does not erase the main memory. The existing data is preserved.

6.2 Option Byte Programming

Option bytes provide a set of user-configurable functions. They include 12 configurable bytes and their corresponding complement bytes. After power-on, the option byte region is loaded into the FMC_OBCS and FMC_WRTPROT0/1 registers. Option byte settings only take effect after they are reloaded into the FMC registers.

Erasing Option Bytes

Option bytes support erase operations. After a successful option byte erase or option byte write operation, the OCF bit in FMC_STS is set. If the OCIE interrupt is enabled, an operation-complete interrupt is triggered.
Erase procedure:

(1) Unlock FLASH and check the BUSYF flag to confirm that no FLASH operation is in progress.

(2) Check BUSYF again to confirm that no FLASH operation is in progress.

(3) Unlock the option byte region and wait until the OBWEN bit is set.

(4) Set the OBE bit.

(5) Set the STA bit.

(6) Wait until BUSYF is cleared, then perform read-back verification on the erased address.

Writing Option Bytes

All 12 configurable option bytes support write operations.

Programming procedure:

(1) Check the BUSYF flag to confirm that no FLASH operation is in progress.

(2) Unlock the option byte region and wait until OBWEN is set.

(3) Set the OBP bit.

(4) Write the programming start address to FMC_ADDR. The address must be double-word aligned.

(5) Configure PROGLEN. Then write the data sequentially to FLASH_PROG_DATAx. The hardware writes the complement data automatically.

(6) Set the STA bit.

(7) Wait until BUSYF is cleared, then perform read-back verification on the written address.

AUTOSAR Fls Write Option Byte Execution Flow:

In AUTOSAR Fls, there is no separate option byte erase command. The interface function Fls_WriteOptionByte(SourceAddressPtr) executes the complete option byte flow. To erase option bytes, program the default values. The chip must be power-cycled after this operation for the new configuration to take effect.

Application Code Example:

```
uint32_t g_OptionByteEraseBuf[6] = {
0xFFFF55AA, // [0] 0x1FFFF800–0x1FFFF803: READPROT=0xAA, no read protection; UOB=0xFF, default; complement placeholder
0xFFFFFFFF, // [1] 0x1FFFF804-0x1FFFF807: Data0/Data1=0xFF, default
0xFFFFFFFF, // [2] 0x1FFFF808-0x1FFFF80B: WRP0/WRP1=0xFF, no write protection
0xFFFFFFFF, // [3] 0x1FFFF80C-0x1FFFF80F: WRP3/WRP2=0xFF, no write protection
0xFFFFFFFF, // [4] 0x1FFFF810-0x1FFFF813: WRP5/WRP4=0xFF, no write protection
0xFFFFFFFF // [5] 0x1FFFF814-0x1FFFF817: WRP7/WRP6=0xFF, no write protection
};

Fls_WriteOptionByte(g_OptionByteEraseBuf); // Trigger erase and restore default values
```

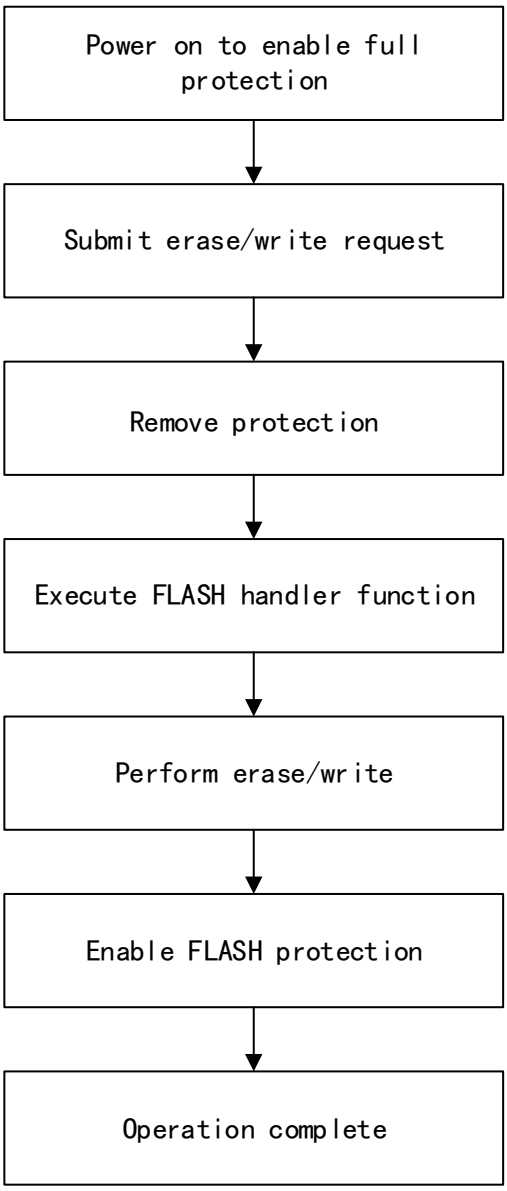
7 FLASH Precautions

7.1 Recommendation 1

Keep FLASH fully protected when no FLASH operation is running. Remove protection before erase or write operations, and restore full protection immediately after the operation completes. For example, in an application that supports firmware upgrade, configure both the BOOT region and APP region, meaning the entire FLASH, as fully protected at power-on. During the upgrade, protect only the BOOT region. After the upgrade completes, restore full FLASH protection.

FLASH Protection Execution Flow

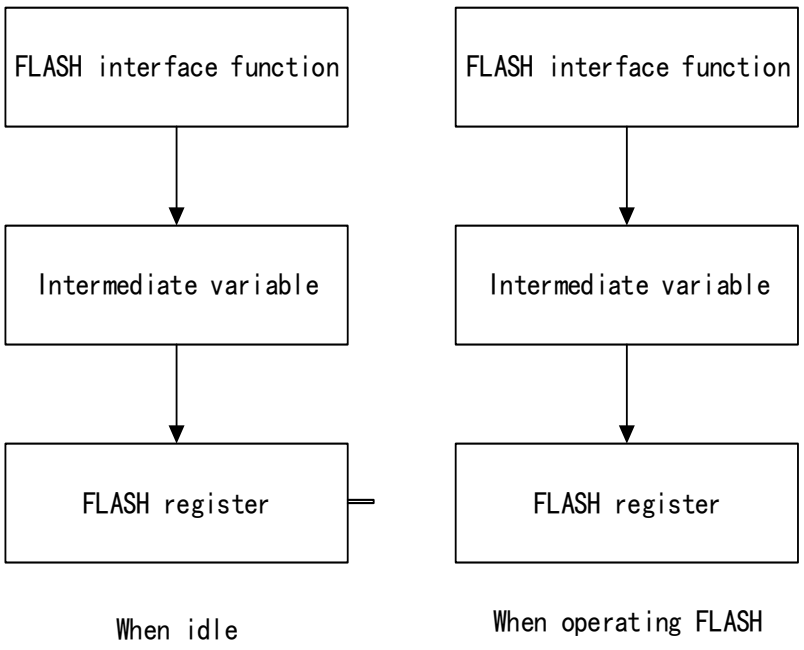
Figure 1 FLASH Protection Flow Chart



7.2 Recommendation 2

Modify the low-level driver and coordinate with the application layer so the original FLASH low-level interface functions do not access registers directly. Instead, use a pointer variable to reference the registers. By default, this pointer should not point to any register. Only when a FLASH operation is required should the pointer be set to the FLASH register address.

Figure 2 Flash Register Operation Flow



7.3 Recommendation 3

In upgrade scenarios, dynamically load the FLASH erase and write functions into RAM and execute them from RAM. For applications with firmware upgrade support, do not keep FLASH erase/write interface functions in ROM or FLASH. During an upgrade, use CAN diagnostic messages to transfer the FLASH erase/write functions into RAM for execution. After the upgrade completes, clear the functions from RAM. This prevents static FLASH operation functions stored in ROM from being accidentally called.

8 Revision History

Table 4 Document Revision History

Date	Version	Revision History
May, 2026	1.0	● Initial release

Statement

This document is formulated and published by Geehy Semiconductor Co., Ltd. (hereinafter referred to as “Geehy”). The contents in this document are protected by laws and regulations of trademark, copyright and software copyright. Geehy reserves the right to make corrections and modifications to this document at any time. Read this document carefully before using Geehy products. Once you use the Geehy product, it means that you (hereinafter referred to as the “users”) have known and accepted all the contents of this document. Users shall use the Geehy product in accordance with relevant laws and regulations and the requirements of this document.

1. Ownership

This document can only be used in connection with the corresponding chip products or software products provided by Geehy. Without the prior permission of Geehy, no unit or individual may copy, transcribe, modify, edit or disseminate all or part of the contents of this document for any reason or in any form.

The “极海” or “Geehy” words or graphics with “®” or “™” in this document are trademarks of Geehy. Other product or service names displayed on Geehy products are the property of their respective owners.

2. No Intellectual Property License

Geehy owns all rights, ownership and intellectual property rights involved in this document.

Geehy shall not be deemed to grant the license or right of any intellectual property to users explicitly or implicitly due to the sale or distribution of Geehy products or this document.

If any third party’s products, services or intellectual property are involved in this document, it shall not be deemed that Geehy authorizes users to use the aforesaid third party’s products, services or intellectual property. Any information regarding the application of the product, Geehy 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, unless otherwise agreed in sales order or sales contract.

3. Version Update

Users can obtain the latest document of the corresponding models when ordering Geehy products.

If the contents in this document are inconsistent with Geehy products, the agreement in the sales order or the sales contract shall prevail.

4. Information Reliability

The relevant data in this document are obtained from batch test by Geehy Laboratory or cooperative third-party testing organization. However, clerical errors in correction or errors caused by differences in testing environment may occur inevitably. Therefore, users should understand that Geehy does not bear any responsibility for such errors that may occur in this document. The relevant data in this document are only used to guide users as performance parameter reference and do not constitute Geehy's guarantee for any product performance.

Users shall select appropriate Geehy products according to their own needs, and effectively verify and test the applicability of Geehy products to confirm that Geehy products meet their own needs, corresponding standards, safety or other reliability requirements. If losses are caused to users due to user's failure to fully verify and test Geehy products, Geehy will not bear any responsibility.

5. Legality

USERS SHALL ABIDE BY ALL APPLICABLE LOCAL LAWS AND REGULATIONS WHEN USING THIS DOCUMENT AND THE MATCHING GEEHY PRODUCTS. USERS SHALL UNDERSTAND THAT THE PRODUCTS MAY BE RESTRICTED BY THE EXPORT, RE-EXPORT OR OTHER LAWS OF THE COUNTRIES OF THE PRODUCTS SUPPLIERS, GEEHY, GEEHY DISTRIBUTORS AND USERS. USERS (ON BEHALF OR ITSELF, SUBSIDIARIES AND AFFILIATED ENTERPRISES) SHALL AGREE AND PROMISE TO ABIDE BY ALL APPLICABLE LAWS AND REGULATIONS ON THE EXPORT AND RE-EXPORT OF GEEHY PRODUCTS AND/OR TECHNOLOGIES AND DIRECT PRODUCTS.

6. Disclaimer of Warranty

THIS DOCUMENT IS PROVIDED BY GEEHY "AS IS" AND THERE IS NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, TO THE EXTENT PERMITTED BY APPLICABLE LAW.

GEEHY'S PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED FOR USE AS CRITICAL COMPONENTS IN MILITARY, LIFE-SUPPORT, POLLUTION CONTROL, OR HAZARDOUS SUBSTANCES MANAGEMENT SYSTEMS, NOR WHERE FAILURE COULD RESULT IN INJURY, DEATH, PROPERTY OR ENVIRONMENTAL DAMAGE.

IF THE PRODUCT IS NOT LABELED AS "AUTOMOTIVE GRADE," IT SHOULD NOT BE CONSIDERED SUITABLE FOR AUTOMOTIVE APPLICATIONS. GEEHY ASSUMES NO LIABILITY FOR THE USE BEYOND ITS SPECIFICATIONS OR GUIDELINES.

THE USER SHOULD ENSURE THAT THE APPLICATION OF THE PRODUCTS COMPLIES WITH ALL RELEVANT STANDARDS, INCLUDING BUT NOT LIMITED TO SAFETY, INFORMATION SECURITY, AND ENVIRONMENTAL REQUIREMENTS. THE USER ASSUMES FULL RESPONSIBILITY FOR THE SELECTION AND USE OF GEEHY PRODUCTS. GEEHY WILL BEAR NO RESPONSIBILITY FOR ANY DISPUTES ARISING FROM THE SUBSEQUENT DESIGN OR USE BY USERS.

7. Limitation of Liability

IN NO EVENT, UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL GEEHY OR ANY OTHER PARTY WHO PROVIDES THE DOCUMENT AND PRODUCTS "AS IS", BE LIABLE FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, DIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE DOCUMENT AND PRODUCTS (INCLUDING BUT NOT LIMITED TO LOSSES OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY USERS OR THIRD PARTIES). THIS COVERS POTENTIAL DAMAGES TO PERSONAL SAFETY, PROPERTY, OR THE ENVIRONMENT, FOR WHICH GEEHY WILL NOT BE RESPONSIBLE.

8. Scope of Application

The information in this document replaces the information provided in all previous versions of the document.