GigaDevice Semiconductor Inc.

Transplantation of IEC60730 certification library based on RISC-V core in IAR environment

Application Note
AN071

Table of Contents

Table of Contents 2
List of Figures 3
List of Tables 4
1. Introduction 5
2. Migrating IEC60730 certification library based on IAR environment and RISC-V core 6
2.1. IEC60730 certification library migration platform 6
2.2. Create new IAR project 7
2.3. Modify cstartup.s file 8
2.4. Modify gd32vf103_test_cpu_prerun_IAR.s and gd32vf103_test_cpu_run_IAR.s 8
2.5. Modify scatter-loading files 11
2.6. Modify RAM detection code 15
3. Code test 16
4. Revision history 17

List of Figures

Figure 2-1. IEC60730 certification library source code ... 6
Figure 2-2. IAR project directory ... 7
Figure 2-3. Compile window prompt ... 11
Figure 2-4. test_fail_rest() function scatter loading ... 12
Figure 3-1. Super side output print information ... 16

List of Tables

Table 2-1. Add test_prerun() function 8
Table 2-2. gd32vf103_test_cpu_prerun_IAR.s code 9
Table 2-3. gd32vf103_test_cpu_run_IAR.s code 10
Table 2-4. Scatter loading code 12
Table 2-5. SP read function 15
Table 2-6. Start address data protection code 15
Table 4-1. Revision history 17

1. Introduction

At present, the GD32 MCU based on RISC-V core supports two development environments, Eclipse and IAR. The IEC60730 certification library based on the Eclipse environment has been developed. In order to improve the diversity of the IEC60730 certification library, it is necessary to transplant the IEC60730 certification library in the IAR environment. This article introduces the problems and corresponding solutions when transplanting the IEC60730 certification library in the IAR environment.

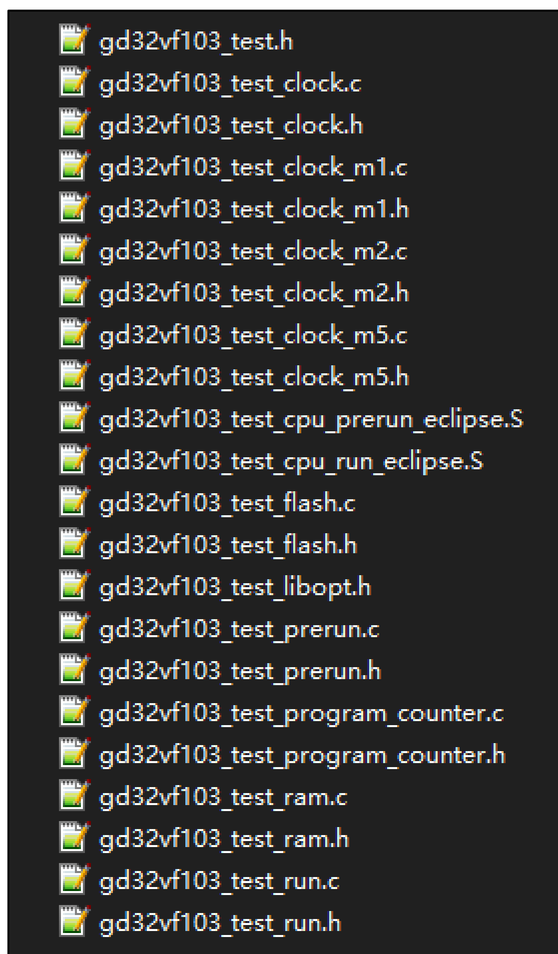
2. Migrating IEC60730 certification library based on IAR environment and RISC-V core

2.1. IEC60730 certification library migration platform

This article transplants the IEC60730 certification library based on the GD32VF103V-EVAL V1.0 development board, and realizes the functional detection of different MCU modules (CLOCK, CPU, FLASH, RAM, Watchdog) through the certification program.

IAR environment transplant IEC60730 certification library project based on IEC60730 certification library project in eclipse environment, mainly including the following files such as shown in ***Figure 2-1. IEC60730 certification library source code***

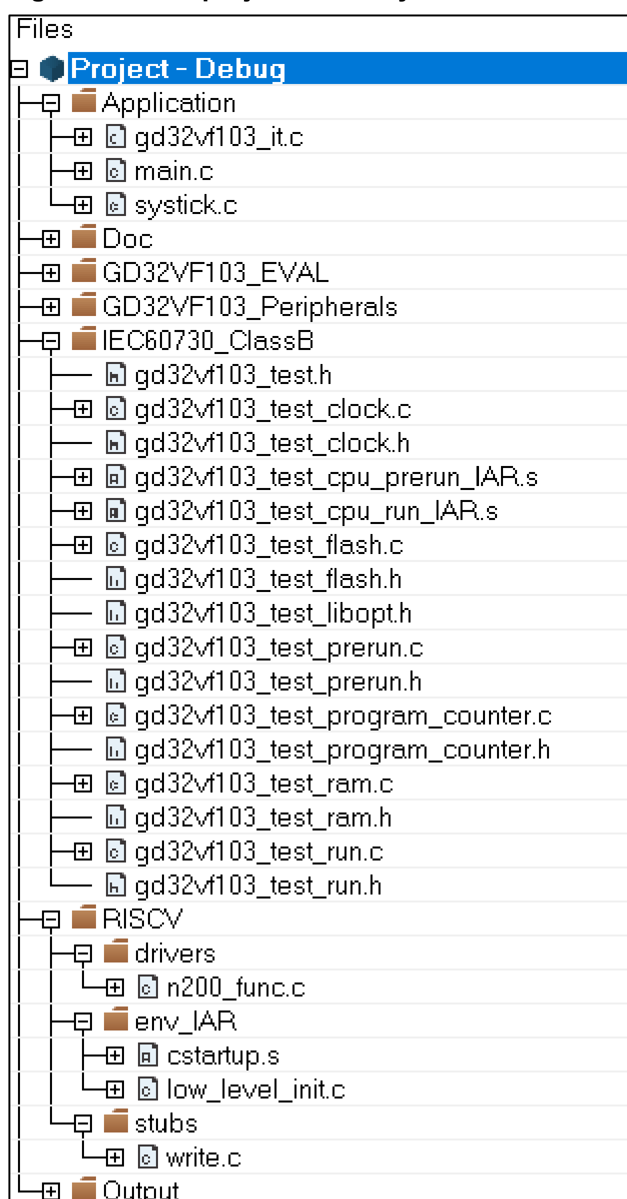
Figure 2-1. IEC60730 certification library source code



2.2. Create new IAR project

Add the EWRISC-V folder under the folder of the IEC60730 project containing the Eclipse environment, then create a new project in IAR (IAR EW for RISC-V 1.40.1), and add the required files for the project. After the files are added, the project needs to be added. file directory as shown in *Figure 2-2. IAR project directory*. The files gd32vf103_test_cpu_prerun_IAR.s and gd32f30x_test_cpu_run_IAR.s are directly obtained by modifying the names of the files gd32vf103_test_cpu_prerun_eclipse.S and gd32vf103_test_cpu_run_eclipse.S in *Figure 2-1. IEC60730 certification library source code*.

Figure 2-2. IAR project directory



2.3. Modify cstartup.s file

The function test in IEC60730 is divided into two stages, the system startup self-test and the running self-test, so it is necessary to modify the cstartup.s code, and call the test_prerun() function before the program runs to the main() function, so as to complete the system startup self-test, The code after adding is shown in ***Table 2-1. Add test_prerun() function***.

It should be noted that the __iar_data_init2() function should be called again after calling the test_prerun() function. If it is not called during the execution of the test_prerun() function, the initialization of the data by the __iar_data_init2() function will be destroyed, resulting in the main() function unable to output print information.

Table 2-1. Add test_prerun() function

```
EXTERN    test_prerun
CfiCall test_prerun
call       test_prerun

beq        a0, zero, ?cstart_call_main

// Reinitialize the data segment
EXTERN    __iar_data_init2
CfiCall __iar_data_init2
call       __iar_data_init2
```

2.4. Modify gd32vf103_test_cpu_prerun_IAR.s and gd32vf103_test_cpu_run_IAR.s

After completing the modification in Section 2.3, compile the project. The gd32vf103_test_cpu_prerun_IAR.s and gd32vf103_test_cpu_run_IAR.s files will report an error, prompting a syntax error, and the error command is the end return command: ret. pass. In the follow-up debugging, it was found that if only the "ret" instruction or the "end" instruction was added, the program could not run normally. The two instructions should be used together, that is, "ret" first and then "end".

Comparing the assembly files of the CPU detection IAR environment under the ARM core, it is necessary to add the relevant code to save the general-purpose registers in the gd32vf103_test_cpu_prerun_IAR.s and gd32vf103_test_cpu_run_IAR.s files, otherwise, the subsequent code will fail to run. After the code is modified as shown in ***Table 2-2. gd32vf103_test_cpu_prerun_IAR.s code*** and ***Table 2-3. gd32vf103_test_cpu_run_IAR.s code***,codes marked in red are modified and added codes.

Table 2-2. gd32vf103_test_cpu_prerun_IAR.s code

```
        SECTION factor_def:CODE:NOROOT(2)

        PUBLIC test_cpu_prerun
        IMPORT test_fail_reset

GENERAL_FACTOR1   EQU   0xAAAAAAAA
GENERAL_FACTOR2   EQU   ~GENERAL_FACTOR1

SP_FACTOR1          EQU   0xAAAAAAA8
SP_FACTOR2          EQU   0x55555554

LOG_REGBYTES        EQU 3
REGBYTES            EQU (1 << LOG_REGBYTES)
;/*
;   \brief   cpu test in prerun time
;   \param   none
;   \retval TypeState: ERROR or SUCCESS
;*/
test_cpu_prerun:

        addi sp, sp, -20*REGBYTES
        sw x1,   0*REGBYTES(sp)
        sw x4,   1*REGBYTES(sp)
        sw x5,   2*REGBYTES(sp)
        sw x6,   3*REGBYTES(sp)
        sw x7,   4*REGBYTES(sp)
        sw x10, 5*REGBYTES(sp)
        sw x11, 6*REGBYTES(sp)
        sw x12, 7*REGBYTES(sp)
        sw x13, 8*REGBYTES(sp)
        sw x14, 9*REGBYTES(sp)
        sw x15, 10*REGBYTES(sp)
        ……
        ……
        ……
        lw x1,   0*REGBYTES(sp)
        lw x6,   2*REGBYTES(sp)
        lw x7,   3*REGBYTES(sp)
        lw x10, 4*REGBYTES(sp)
        lw x11, 5*REGBYTES(sp)
        lw x12, 6*REGBYTES(sp)
        lw x13, 7*REGBYTES(sp)
        lw x14, 8*REGBYTES(sp)
```

```
        lw x15, 9*REGBYTES(sp)
        addi sp, sp, 20*REGBYTES

        li      t0,1
        mv      a0,t0                   // SUCCESS = 1

        ret
        end
```

Table 2-3. gd32vf103_test_cpu_run_IAR.s code

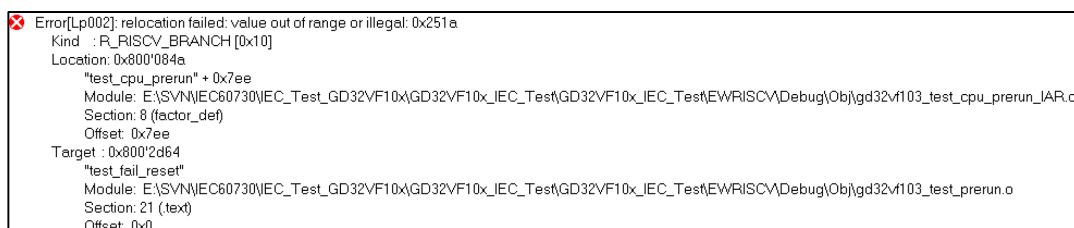
```
        SECTION factor_def:CODE:NOROOT(2)

        PUBLIC test_cpu_run
        IMPORT test_fail_reset

GENERAL_FACTOR1   EQU   0xAAAAAAAA
GENERAL_FACTOR2   EQU   ~GENERAL_FACTOR1

LOG_REGBYTES        EQU 3
REGBYTES            EQU (1 << LOG_REGBYTES)
;/*
;   \brief   cpu test in prerun time
;   \param   none
;   \retval TypeState: ERROR or SUCCESS
;*/
test_cpu_prerun:

        addi sp, sp, -20*REGBYTES
        sw x1,    0*REGBYTES(sp)
        sw x4,    1*REGBYTES(sp)
        sw x5,    2*REGBYTES(sp)
        sw x6,    3*REGBYTES(sp)
        sw x7,    4*REGBYTES(sp)
        sw x10, 5*REGBYTES(sp)
        sw x11, 6*REGBYTES(sp)
        sw x12, 7*REGBYTES(sp)
        sw x13, 8*REGBYTES(sp)
        sw x14, 9*REGBYTES(sp)
        sw x15, 10*REGBYTES(sp)
        ……
        ……
        ……
        lw x1,   0*REGBYTES(sp)
```

```
lw x6,   2*REGBYTES(sp)
lw x7,   3*REGBYTES(sp)
lw x10, 4*REGBYTES(sp)
lw x11, 5*REGBYTES(sp)
lw x12, 6*REGBYTES(sp)
lw x13, 7*REGBYTES(sp)
lw x14, 8*REGBYTES(sp)
lw x15, 9*REGBYTES(sp)
addi sp, sp, 20*REGBYTES

li      t0,1
mv      a0,t0                   // SUCCESS = 1

ret
end
```

2.5. Modify scatter-loading files

Before compiling the project, it is necessary to compare the loading files of other ARM core series and modify the scattered loading files required for this project. The standard scattered loading files can be found in the IAR installation directory:"…\riscv\config\linker\GigaDevice". After completing the above steps to compile, the following error will appear as shown in ***Figure 2-3. Compile window prompt***.The reason for the error is that when the B-type conditional jump instruction in the RISC-V assembly instruction jumps, the addressing range of the PC is (+ / -) 4KB. Therefore, the solution is to use scatter loading to combine the test_cpu_prerun() function, test_cpu_run() function and The test_fail_rest() function is located within 4KB of the FLASH space.

Figure 2-3. Compile window prompt



For the above scatter loading, the gd32vf103_test_cpu_prerun_IAR.o, gd32vf103_test_cpu_run_IAR.o and gd32vf103_test_prerun.o files containing the test_cpu_prerun() function, test_cpu_run() function and test_fail_rest() function are first scatter-loaded into the 4KB space, but since the files containing the test_fail_rest() function The space required for the .o file gd32vf103_test_prerun.o is large, resulting in more than 4KB of space for the three .o files. Querying the Help manual of IAR, you can allocate the

storage location to the functions in the .c file separately, as shown in ***Figure 2-4. test_fail_rest() function scatter loading***.

Figure 2-4. test_fail_rest() function scatter loading

```
*/
void test_fail_reset(void) @ ".TEST_FAIL_RESET"
{
```

After the above processing, perform scatter loading for gd32vf103_test_cpu_prerun_IAR.o, gd32vf103_test_cpu_run_IAR.o and . TEST_FAIL_RESET, and the required space does not exceed 4KB. After completing the comparison modification and function space position modification, the scatter loading code is shown in ***Table 2-4. Scatter loading code***.

Table 2-4. Scatter loading code

```
///////////////////////////////////////////////////////////////////////////////
// RISC-V ilink configuration file.
//

define exported symbol _link_file_version_2 = 1;
define exported symbol _auto_vector_setup = 1;
define exported symbol _max_vector = 96;
define exported symbol _CLINT = 1;

define memory mem with size = 4G;

/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__     = 0x0801FFFF;

define symbol __ICFEDIT_region_ROM1_start__ = 0x08004000;
define symbol __ICFEDIT_region_ROM1_end__     = 0x08004fff;


define symbol __ICFEDIT_region_RAM_start__ = 0x200000B0;
define symbol __ICFEDIT_region_RAM_end__     = 0x20007FFF;
define symbol __ICFEDIT_region_IECTEST_PARAM_start__    = 0x20000040;
define symbol __ICFEDIT_region_IECTEST_PARAM_end__        = 0x200000B0;

/*Sizes*/
define symbol __ICFEDIT_size_stack_ov_test__        = 0x18;
/**** End of ICF editor section. ###ICF###*/

export symbol   __ICFEDIT_region_ROM_start__;
export symbol   __ICFEDIT_region_ROM_end__;
export symbol   __ICFEDIT_region_RAM_start__;
```

```
export symbol   __ICFEDIT_region_RAM_end__;
export symbol   __ICFEDIT_region_IECTEST_PARAM_start__;
export symbol   __ICFEDIT_region_IECTEST_PARAM_end__;



define region ROM_region32 = mem:[from __ICFEDIT_region_ROM_start__ to __ICFEDIT_region_ROM_end__];
define region RAM_region32 = mem:[from __ICFEDIT_region_IECTEST_PARAM_end__ to __ICFEDIT_region_RAM_end__];
define region ROM1_region32 = mem:[from __ICFEDIT_region_ROM1_start__ to __ICFEDIT_region_ROM1_end__];

/*-Symbols-*/
define symbol __region_RAM_RUN_BUF_start__   = 0x20000004;
define symbol __region_RAM_RUN_PTR__satrt__ = 0x20000030;
define symbol __region_IEC_TEST_RAM_start__   = 0x20000040;

/*-Memory Regions-*/
define region RAM_BUF_region = mem:[from __region_RAM_RUN_BUF_start__ to 0x2000002F];
define region RAM_PTR_region = mem:[from __region_RAM_RUN_PTR__satrt__ to 0x2000003F];
define region IEC_TEST_VAR_region    = mem:[from __region_IEC_TEST_RAM_start__ to 0x200000AF];
/**** End of ICF editor Regions. ###ICF###*/

initialize by copy { rw };

do not initialize   { section *.noinit,
                      section   STACK_OV_TEST,
                      section RAM_RUN_BUF,
                      section RAM_RUN_PTR,
                      section IEC_TEST_RAM};
place in ROM1_region32           { object gd32vf103_test_cpu_prerun_IAR.o,
                                   section .TEST_FAIL_RESET,
                                   object          gd32vf103_test_cpu_run_IAR.o};          //Make test_cpu_prerun(), test_cpu_run() and test_fail_reset() no more than 0x1000 in the flash by scatter loading

define block CSTACK with alignment = 16, size = CSTACK_SIZE { };
define block HEAP    with alignment = 16, size = HEAP_SIZE    { };
define block STACK_OV_TEST   with alignment = 8, size = __ICFEDIT_size_stack_ov_test__ { };

define block MVECTOR with alignment = 128, size = _max_vector*4 { ro section .mintvec };
```

```
place in IEC_TEST_VAR_region
                              { rw data section IEC_TEST_RAM    };

place in RAM_BUF_region
                              { rw data section RAM_RUN_BUF    };

place in RAM_PTR_region
                              { rw data section RAM_RUN_PTR    };

if (isdefinedsymbol(_uses_clic))
{
   define block MINTERRUPT with alignment = 128 { ro section .mtext };
   define block MINTERRUPTS { block MVECTOR,
                                         block MINTERRUPT };
}
else
{
   define block MINTERRUPTS with maximum size =    64k { ro section .mtext,
                                                             midway block MVECTOR };
}

define block RW_DATA with static base GPREL { rw data };
keep { symbol __iar_cstart_init_gp }; // defined in cstartup.s
keep { ro section .alias.hwreset };

"CSTARTUP32" : place at start of ROM_region32 { ro section .alias.hwreset,
                                                          ro section .cstartup };


"ROM32":place in ROM_region32              { ro,
                                               block MINTERRUPTS }
                              except       {object gd32vf103_test_cpu_prerun_IAR.o,
                                               section .TEST_FAIL_RESET,
                                               object gd32vf103_test_cpu_run_IAR.o};
   place at end of ROM_region32           { ro section .checksum };

"RAM32":place in RAM_region32             { block RW_DATA,
                                              block HEAP,
                                              block CSTACK,
                                              block STACK_OV_TEST };
```

2.6. Modify RAM detection code

Since RAM detection will destroy the stack content, it is necessary to save the stack before operating the RAM, and restore the stack after the detection is completed. Compared with the ARM core detection code, the SP is obtained first, and then the stack content is saved. The compilation reflects the incompatibility of the SP read function under the two architectures, and the SP read function needs to be reimplemented. Check the RISC-V core instructions, you can get SP through the mscratch register. First, write the SP into the mscratch register through the inline function, and then read the value of the mscratch register, so as to realize the reading of the SP. The code implementation is shown in ***Table 2-5. SP read function***.

Table 2-5. SP read function

```
void write_sp(void){     asm("csrrw sp, mscratch,sp");}
void read_sp(void){     asm("csrrw sp, mscratch,sp");}
write_sp();
ptr_stack = (uint32_t *)read_csr(CSR_MSCRATCH)+8;
read_sp();
```

After the stack content is saved, compile and run the program, the program execution jumps normally, but the printf() function cannot print normally, single-step debugging, and modify the RAM detection start address to 0x20000140, it can print normally, indicating that starting from the RAM The starting address stores important information. Therefore, before the RAM detection, not only the stack but also the data of the starting address need to be saved. For this reason, the following code is added to the RAM detection code, as shown in ***Table 2-6. Start address data protection code***.

Table 2-6. Start address data protection code

```
ptr_stack = (uint32_t *)(RAM_START+128);
/* store the value of   RAM (0x2000 0000 - 0x2000 0200) into the end of RAM */
for(i = 128; i != 0; i--) {
    *(__IO uint32_t *)(RAM_END + i+384) = *ptr_stack;
    ptr_stack--;
}
/* restore the value of RAM (0x2000 0000 - 0x2000 0200) from the end of RAM */
ptr_stack = (uint32_t *)(RAM_START+128);
    for(i = 128; i != 0; i--) {
    *ptr_stack = *(__IO uint32_t *)(RAM_END + i + 384);
     ptr_stack--;
}
```

3. Code test

Compile and run the project, and it can be seen from the printing information output by the super-side printing window that the program is running normally and the functions of each module are being tested.

Figure 3-1. Super side output print information

```
>>>>>>>>>>>>>>  IEC60730 Test Board Init Success  <<<<<<<<<<<<<<


CPU Test(PreRun) Success!

... Power reset or software reset, next step --> FWDGT reset test ...


>>>>>>>>>>>>>>  IEC60730 Test Board Init Success  <<<<<<<<<<<<<<


CPU Test(PreRun) Success!

FWDGT reset
... FWDGT reset test OK, next step --> WWDGT reset test ...


>>>>>>>>>>>>>>  IEC60730 Test Board Init Success  <<<<<<<<<<<<<<


CPU Test(PreRun) Success!

FWDGT reset
WWDGT reset

... WWDGT reset test OK, WDGT test completed ...


Full RAM Test Success!


FLASH CRC32 Test(PreRun) Success!

Clock Frequency Test Success!

Program counter test(PreRun) Success!


**************************** Main program starts
****************************


FLASH CRC(Run-Time) Test running! Next Address --> 0x08000080
```

4. Revision history

Table 4-1. Revision history

Revision No.	Dscription	Date
1.0	Initial Release	Sep.20 2022

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 under the intellectual property laws and treaties of the People’s Republic of China and other jurisdictions worldwide. The Company reserves all rights under such laws and treaties and does not grant any license under its patents, copyrights, trademarks, or other intellectual property rights. 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.

The Company makes no warranty of any kind, express or implied, with regard to this document or any Product, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The Company does not assume any liability arising out of the application or use of any Product described in this document. Any information provided in this document is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Except for customized products which has been expressly identified in the applicable agreement, the Products are designed, developed, and/or manufactured for ordinary business, industrial, personal, and/or household applications only. The Products are not designed, intended, or authorized for use as components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, atomic energy control instruments, combustion control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or Product could cause personal injury, death, property or environmental damage ("Unintended Uses"). Customers shall take any and all actions to ensure using and selling the Products in accordance with the applicable laws and regulations. The Company is not liable, in whole or in part, and customers shall and hereby do release the Company as well as it’s suppliers and/or distributors from any claim, damage, or other liability arising from or related to all Unintended Uses of the Products. Customers shall indemnify and hold the Company as well as it’s 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 Products.

Information in this document is provided solely in connection with the Products. The Company reserves the right to make changes, corrections, modifications or improvements to this document and Products and services described herein at any time, without notice.

© 2022 GigaDevice – All rights reserved