GigaDevice Semiconductor Inc.

Arm® Cortex®-M3/4/23/33 32-bit MCU

Application Note
AN039

Table of Contents

List of Figures

List of Tables

1. Introduction

In the IEC60730 self-check, it requires self-check on the on-chip flash of the mcu. In order to realize the automatic calculation and addition of the CRC value, it is necessary to add a CRC check step in the IDE. To this end, this application note describes how to add a CRC check batch method in the eclipse environment. The process is described as follows.

2. CRC check batch processing

In the eclipse environment, the function of automatically calculating CRC is not provided. In order to complete the flash self-check required by the IEC60730 standard, the CRC value is usually pre-calculated and added to the link file manually. This method is complicated to operate. For this reason, by adding Batch processing steps to complete the automatic calculation and addition of the CRC value.

2.1. Add CRC value batch processing required files

1. First download the Srecord tool, create a new bin folder in the project directory, and copy srec_cat.exe, srec_cmp.exe and srec_info.exe to this folder.
2. Add the gen_crc.bat and gd32e10x_flash.ld files in the project directory. Here we take the gd32e10x series chips as an example. The gen_crc.bat file is used to call the Srecord tool to calculate the CRC value and store the value at the end of the .hex file to facilitate the self-check of the entire flash and print it in the build window. CRC value calculated by Srecord tool. The command code of the file is as follows:

```
SET MAP_FILE=gd32e10x_iec_test.map
::-----------------get CRC address information line
SET TMP_FILE=crc_temp.txt
FINDSTR /R /C:"^ *.check_sum" %MAP_FILE%>%TMP_FILE%
SET /p crc_search=<%TMP_FILE%
DEL %TMP_FILE%
::-----------------CRC address
for /f "tokens=1 delims=(" %%a in ("%crc_search%") do set crc_search=%%a
SET crc_search=%crc_search:.check_sum=%
for /f "tokens=1 delims= " %%a in ("%crc_search%") do set CRC_ADDR=%%a


SET /a CRC_ADDR_END=%CRC_ADDR%+4


::-----------------choose CRC32 or CRC16
FINDSTR /R /C:"^ .*crc_block_data_calculate" %MAP_FILE% > nul &&
call :OK||call :NO
goto :eof

:OK
::--------------------CRC32
..\bin\srec_cat.exe Project.hex -intel -crop 0x08000000 %CRC_ADDR% -fill 0xff
0x08000000 %CRC_ADDR% -stm32-l-e %CRC_ADDR% -o Project_checked.hex -
```

```
intel
..\bin\srec_cat.exe   Project.hex   -intel      -crop   0x08000000   %CRC_ADDR%
Project_checked.hex  -intel    -crop  %CRC_ADDR%  %CRC_ADDR_END%  -o
Project.hex -intel
..\bin\srec_cat.exe  Project.hex  -intel   -crop  %CRC_ADDR%  %CRC_ADDR_END%
-o   -hex-dump
del Project_checked.hex
goto :eof

:NO
::-------------------CRC16
..\bin\srec_cat.exe  Project.hex  -intel  -crop  0x08000000  %CRC_ADDR%   -fill  0xff
0x08000000 %CRC_ADDR%   -crc16-l-e %CRC_ADDR% -POLYnomial ccitt -XMODEM
-o   Project_checked.hex -intel
..\bin\srec_cat.exe   Project.hex   -intel      -crop   0x08000000   %CRC_ADDR%
Project_checked.hex  -intel    -crop  %CRC_ADDR%  %CRC_ADDR_END%    -o
Project.hex -intel
..\bin\srec_cat.exe  Project.hex  -intel   -crop  %CRC_ADDR%  %CRC_ADDR_END%
-o   -hex-dump
del Project_checked.hex
goto :eof

exit
```

The gd32e10x_flash.ld file defines the loading address of each program segment and variable. Through the code shown below, the CRC value (CHECKSUM) is fixed at the end of the FLASH space.

```

ENTRY(Reset_Handler)

/* end of Stack */
_estack = 0x20008000;    /*ram size*/

/* memory map */
MEMORY
{
   FLASH (rx)          : ORIGIN = 0x08000000, LENGTH = 128K       /*flash size*/
   iec_test (wxa!ri) : ORIGIN = 0x20000000, LENGTH = 0xB0
   RAM (xrw)           : ORIGIN = 0x200000B0, LENGTH = 0x7F50    /*32K*/
   flash_end (rxai!w) : ORIGIN = 0x0801FFC0, LENGTH = 0x40 /*flash size - 0x40*/
}

SECTIONS
```

```
{
  __stack_size = DEFINED(__stack_size) ? __stack_size : 2K;

  .isr_vector :
  {
    . = ALIGN(4);
    KEEP(*(.isr_vector))
    . = ALIGN(4);
  } >FLASH

  .text :
  {
    . = ALIGN(4);
    *(.text)
    *(.text*)
    *(.glue_7)
    *(.glue_7t)
    *(.eh_frame)

    KEEP (*(.init))
    KEEP (*(.fini))

    . = ALIGN(4);
    /* the symbol '_etext' will be defined at the end of code section */
    _etext = .;
  } >FLASH

  .rodata :
  {
    . = ALIGN(4);
    *(.rodata)
    *(.rodata*)
    . = ALIGN(4);
  } >FLASH

   .ARM.extab :
  {
     *(.ARM.extab* .gnu.linkonce.armextab.*)
  } >FLASH

    .ARM : {
    __exidx_start = .;
      *(.ARM.exidx*)
      __exidx_end = .;
    } >FLASH

  .ARM.attributes : { *(.ARM.attributes) } > FLASH

  .preinit_array     :
  {
    PROVIDE_HIDDEN (__preinit_array_start = .);
    KEEP (*(.preinit_array*))
    PROVIDE_HIDDEN (__preinit_array_end = .);
  } >FLASH

  .init_array :
  {
    PROVIDE_HIDDEN (__init_array_start = .);
    KEEP (*(SORT(.init_array.*)))
    KEEP (*(.init_array*))
    PROVIDE_HIDDEN (__init_array_end = .);
  } >FLASH

  .fini_array :
  {
    PROVIDE_HIDDEN (__fini_array_start = .);
    KEEP (*(.fini_array*))
    KEEP (*(SORT(.fini_array.*)))
    PROVIDE_HIDDEN (__fini_array_end = .);
  } >FLASH

  ._iec_classb(NOLOAD) :
  {
    . = ALIGN(4);
    *(.ram_run_buf)
    *(.ram_run_buf*)
    *(.ram_run_ptr)
    *(.ram_run_ptr*)

    . = ALIGN(4);
    PROVIDE(_iec_start = . );

    *(.iec_test_ram)
    *(.iec_test_ram*)

    . = ALIGN(4);
  PROVIDE(_iec_end = . );
} >iec_test AT>iec_test

/* provide some necessary symbols for startup file to initialize data */
_sidata = LOADADDR(.data);
.data :
{
  . = ALIGN(4);
  /* the symbol '_sdata' will be defined at the data section end start */
  _sdata = .;
  *(.data)
  *(.data*)
  . = ALIGN(4);
  /* the symbol '_edata' will be defined at the data section end */
  _edata = .;
} >RAM AT> FLASH

.check_sum          :
{
  . = ALIGN(64);
  PROVIDE( __Check_Sum = . );

  LONG(0x904eae51);
} >flash_end AT>flash_end

. = ALIGN(4);
.bss :
{
  /* the symbol '_sbss' will be defined at the bss section start */
  _sbss = .;
  __bss_start__ = _sbss;
  *(.bss)
  *(.bss*)
  *(COMMON)
  . = ALIGN(4);
  /* the symbol '_ebss' will be defined at the bss section end */
  _ebss = .;
  __bss_end__ = _ebss;
} >RAM

PROVIDE ( end = _ebss );
PROVIDE ( _end = _ebss );
```

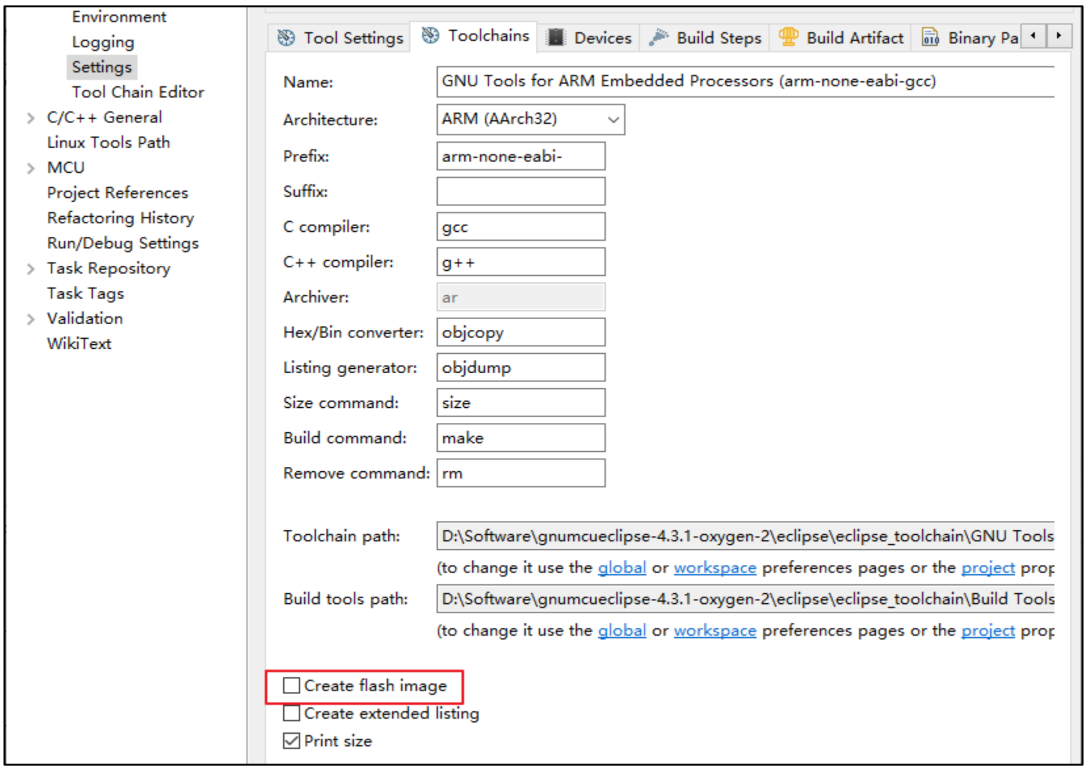
```
  .stack ORIGIN(RAM) + LENGTH(RAM) - __stack_size - 0x18:
  {
    *(.stack_ov_test)
    *(.stack_ov_test*)
    . = 0x18;
    PROVIDE( _heap_end = . );
    . = __stack_size + 0x18;
    PROVIDE( _sp = . );
  }
}

 /* input sections */
GROUP(libgcc.a libc.a libm.a libnosys.a)
```

2.2. Configure batch processing

1. After adding the above two files to the project directory, open the project, right-click the project name, and uncheck Create flash image in Properties->C/C++ Build->Setting->Toolchains. This is to use Batch process to generate executable files, as shown in *Figure 2-1. Uncheck Create flash image*.

Figure 2-1. Uncheck Create flash image



Name: GNU Tools for ARM Embedded Processors (arm-none-eabi-gcc)
Architecture: ARM (AArch32)
Prefix: arm-none-eabi-
Suffix:
C compiler: gcc
C++ compiler: g++
Archiver: ar
Hex/Bin converter: objcopy
Listing generator: objdump
Size command: size
Build command: make
Remove command: rm

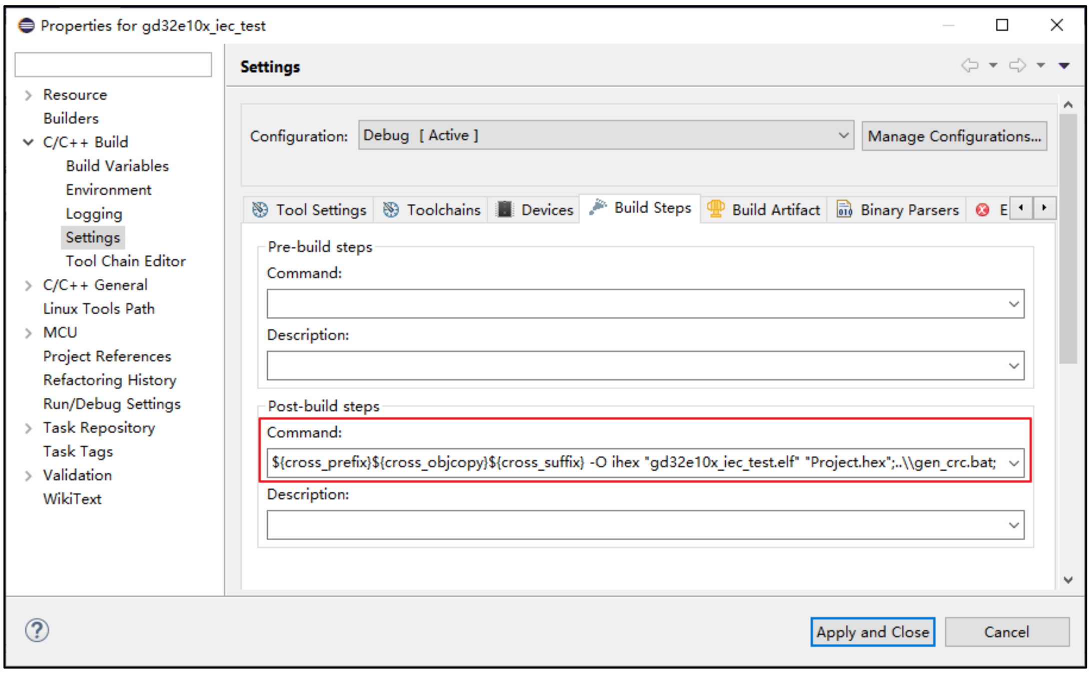
Toolchain path: D:\Software\gnumcueclipse-4.3.1-oxygen-2\eclipse\eclipse_toolchain\GNU Tools
(to change it use the global or workspace preferences pages or the project prop
Build tools path: D:\Software\gnumcueclipse-4.3.1-oxygen-2\eclipse\eclipse_toolchain\Build Tools
(to change it use the global or workspace preferences pages or the project prop

☐ Create flash image
☐ Create extended listing
☑ Print size

2. Right-click the project name->Properties->C/C++build->settings->build steps, and add the following command in Post-build steps:

${cross_prefix}${cross_objcopy}${cross_suffix} -O ihex "gd32f30x_iec_test.elf" "Project.hex";..\\gen_crc.bat;, click apply, this command is to ensure that the operation of converting the .elf file into a .hex file is before the operation of the .bat file command, the setting method is as follows ***Figure 2-2. Post-build steps settings*** shown.

Figure 2-2. Post-build steps settings



Properties for gd32e10x_iec_test

Settings

Configuration: Debug [Active]

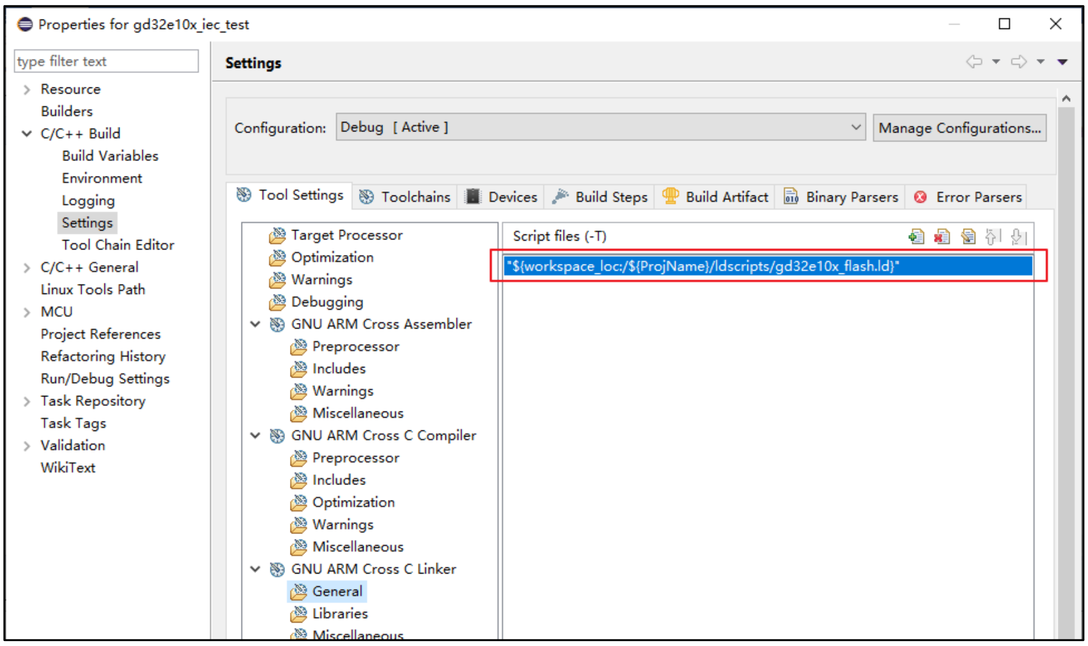
Post-build steps

Command:

```
${cross_prefix}${cross_objcopy}${cross_suffix} -O ihex "gd32e10x_iec_test.elf" "Project.hex";..\\gen_crc.bat;
```

3. Right-click the project name->Properties->C/C++build->settings->GNU ARM Cross C Linker->General, add the link file here, as shown in ***Figure 2-3. Add link file***, after modification Click Apply and Close.

Figure 2-3. Add link file



Properties for gd32e10x_iec_test

Settings

Configuration: Debug [Active]

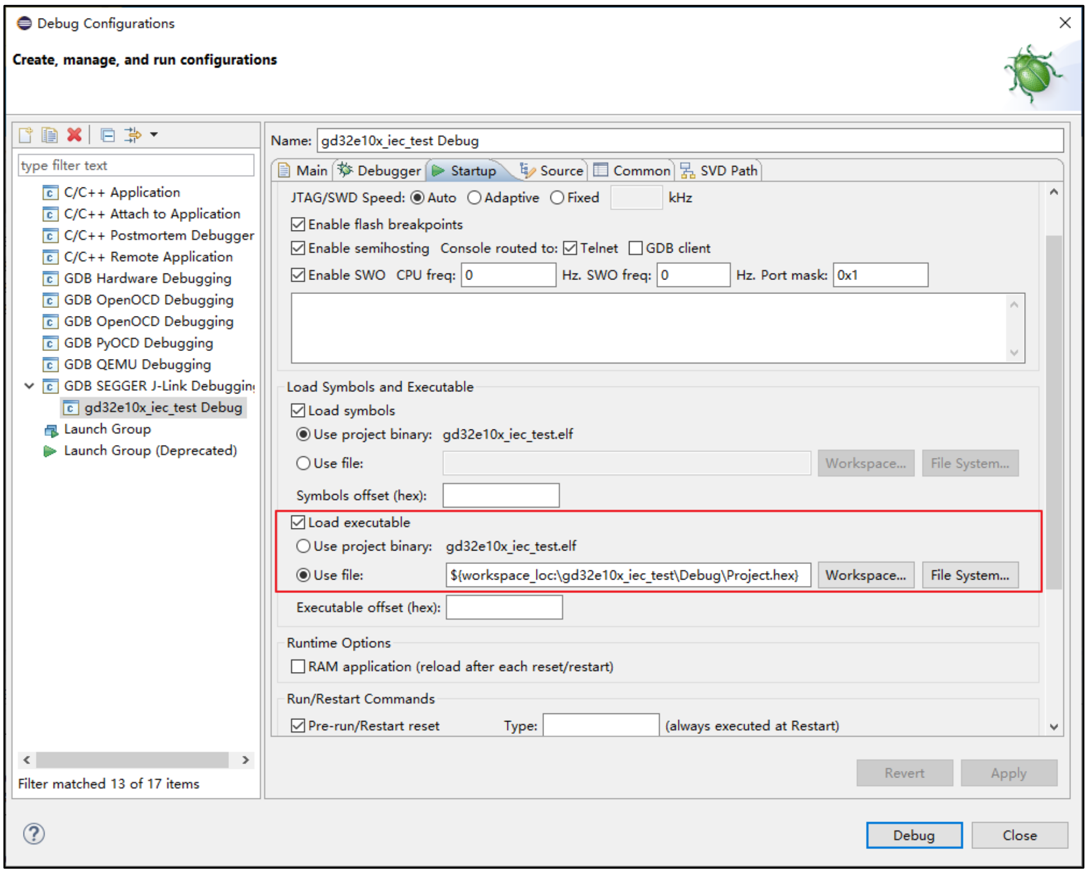
GNU ARM Cross C Linker > General

Script files (-T)

```
"${workspace_loc:/${ProjName}/ldscripts/gd32e10x_flash.ld}"
```

4. When debugging, you need to select the executable file as the Project.hex file generated after batch processing, in the Load executable option of the Run->Debug Configuration->GDB SEGGER J-Link Debugging->gd32e10x_iec_test Debug->Startup interface Choose Use file. As shown in ***Figure 2-4. Select executable file***, click Apply. After the modification, the Project.hex file generated by batch processing is stored in the Debug folder in the workspace.

Figure 2-4. Select executable file



Debug Configurations

Create, manage, and run configurations

type filter text

- C/C++ Application
- C/C++ Attach to Application
- C/C++ Postmortem Debugger
- C/C++ Remote Application
- GDB Hardware Debugging
- GDB OpenOCD Debugging
- GDB OpenOCD Debugging
- GDB PyOCD Debugging
- GDB QEMU Debugging
- GDB SEGGER J-Link Debuggin
  - gd32e10x_iec_test Debug
- Launch Group
- Launch Group (Deprecated)

Filter matched 13 of 17 items

Name: gd32e10x_iec_test Debug

Main | Debugger | Startup | Source | Common | SVD Path

JTAG/SWD Speed: ● Auto ○ Adaptive ○ Fixed kHz

☑ Enable flash breakpoints

☑ Enable semihosting Console routed to: ☑ Telnet ☐ GDB client

☑ Enable SWO CPU freq: 0 Hz. SWO freq: 0 Hz. Port mask: 0x1

Load Symbols and Executable

☑ Load symbols

● Use project binary: gd32e10x_iec_test.elf

○ Use file: Workspace... File System...

Symbols offset (hex):

☑ Load executable

○ Use project binary: gd32e10x_iec_test.elf

● Use file: ${workspace_loc:\gd32e10x_iec_test\Debug\Project.hex} Workspace... File System...

Executable offset (hex):

Runtime Options

☐ RAM application (reload after each reset/restart)

Run/Restart Commands

☑ Pre-run/Restart reset Type: (always executed at Restart)

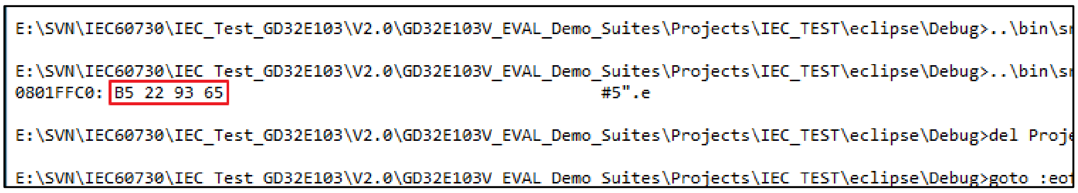
Revert Apply

Debug Close

3. Results

After the configuration is completed, click Compile, and you can observe the build information in the Build Output window, as shown in ***Figure 3-1. Build information***, showing that the CRC value has been stored in the location after 0x0801FFF0; click Debug, and query the address of 0x0801FFF0 in the memory observation window, as shown in ***Figure 3-2. Memory information***, it can be seen that the value in the memory is consistent with the CRC value displayed in the Build Output window, and the CRC value calculation batch has been added successfully.

Figure 3-1. Build information



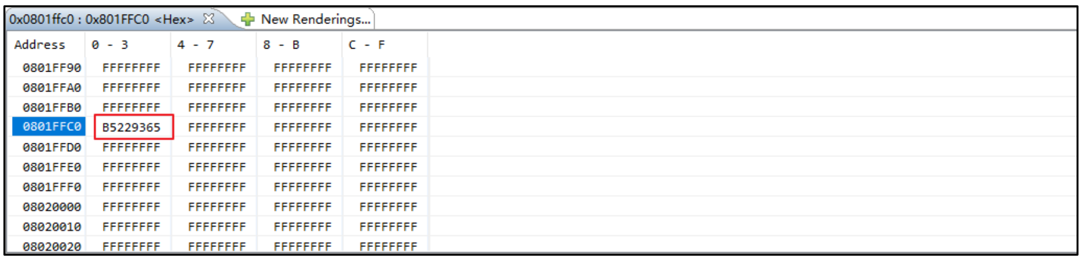
```
E:\SVN\IEC60730\IEC_Test_GD32E103\V2.0\GD32E103V_EVAL_Demo_Suites\Projects\IEC_TEST\eclipse\Debug>..\bin\sr

E:\SVN\IEC60730\IEC_Test_GD32E103\V2.0\GD32E103V_EVAL_Demo_Suites\Projects\IEC_TEST\eclipse\Debug>..\bin\sr
0801FFC0: B5 22 93 65                                       #5".e

E:\SVN\IEC60730\IEC_Test_GD32E103\V2.0\GD32E103V_EVAL_Demo_Suites\Projects\IEC_TEST\eclipse\Debug>del Proje

E:\SVN\IEC60730\IEC_Test_GD32E103\V2.0\GD32E103V_EVAL_Demo_Suites\Projects\IEC_TEST\eclipse\Debug>goto :eof
```

Figure 3-2. Memory information



```
0x0801ffc0 : 0x801FFC0 <Hex>     New Renderings...
Address    0 - 3     4 - 7     8 - B     C - F
0801FF90   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFA0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFB0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFC0   B5229365  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFD0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFE0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
0801FFF0   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
08020000   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
08020010   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
08020020   FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF
```

4. Revision history

Table 4-1.Revision history

Revision No.	Dscription	Date
1.0	Initial Release	Oct.21, 2021