



## **GR551x Fault Trace Module Application Note**

**Version: 1.7**

**Release Date: 2020-12-15**

**Copyright © 2020 Shenzhen Goodix Technology Co., Ltd. All rights reserved.**

Any excerption, backup, modification, translation, transmission or commercial use of this document or any portion of this document, in any form or by any means, without the prior written consent of Shenzhen Goodix Technology Co., Ltd is prohibited.

## **Trademarks and Permissions**

**GOODiX** and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

## **Disclaimer**

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

## **Shenzhen Goodix Technology Co., Ltd.**

Headquarters: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828

FAX: +86-755-33338099

Website: [www.goodix.com](http://www.goodix.com)

## Preface

### Purpose

This document introduces the functionalities, operating mechanisms, and applications of GR551x Fault Trace Module, to help users quickly get started with the Module.

### Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- Technical writer

### Release Notes

This document is the fourth release of *GR551x Fault Trace Module Application Note*, corresponding to GR551x SoC series.

### Revision History

Version	Date	Description
1.3	2020-03-16	Initial release
1.5	2020-05-30	Updated descriptions on the source file <i>cortex_backtrace.c</i> , the initialization module for GR551x Fault Trace Module, and reading fault trace data by using GProgrammer in “Chapter 3 Fault Trace Module in Application”.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-12-15	Updated GRTtoolbox UI figures based on software update.

# Contents

<b>Preface</b> .....	<b>I</b>
<b>1 Introduction</b> .....	<b>1</b>
<b>2 Environment Setup</b> .....	<b>2</b>
2.1 Preparation.....	2
2.2 Hardware Connection.....	2
2.3 Serial Port Settings.....	3
<b>3 Fault Trace Module in Application</b> .....	<b>4</b>
3.1 Import Data to Fault Trace Module to Target Project.....	4
3.1.1 Add the Module.....	4
3.1.2 Enable the Module.....	4
3.1.3 Initialize the Module.....	5
3.2 Read Fault Trace Data.....	5
3.2.1 Read Data via Bluetooth.....	5
3.2.2 Read Data Through GProgrammer.....	8
3.2.3 Read Data by Calling APIs in the Project.....	9
3.3 Operation Demonstration.....	10
<b>4 Module Details</b> .....	<b>14</b>
4.1 HardFault Data Tracing.....	14
4.2 Assert Fault Data Tracing.....	15
4.3 Data Tracing Through Bluetooth.....	17
<b>5 FAQ</b> .....	<b>20</b>
5.1 Why do I Fail to Read Fault Trace Data on GProgrammer?.....	20
5.2 Why do I Fail to Read Fault Trace Data by Calling APIs in the Project?.....	20

# 1 Introduction

GR551x Fault Trace Module aims to help developers identify problems during Bluetooth application development. When GR551x firmware fails to operate normally, GR551x Fault Trace Module can write fault trace data to the Non-Volatile Data Storage (NVDS) in Flash, and export the fault trace data from NVDS, so as to restore the failure scenario and help identify problems.

GR551X Fault Trace Module can write fault trace data to NVDS in the following two scenarios:

- When a HardFault occurs, the Fault Trace Module can write the current values of the internal registers to the NVDS.
- When Assert faults occur, the Fault Trace Module can write the function names, the number of code lines, parameter names, and other relevant information to the NVDS.

Before you get started, it is recommended to refer to the documents listed in [Table 1-1](#).

Table 1-1 Reference documents

Name	Description
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
GR551x Bluetooth Low Energy Stack User Guide	Introduces the layers and basic layer functionalities of a GR551x Bluetooth LE Protocol Stack, and how applications interact with the protocol stack using APIs on the stack.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at <a href="http://www.segger.com/downloads/jlink/UM08001_JLink.pdf">www.segger.com/downloads/jlink/UM08001_JLink.pdf</a> .
Keil User Guide	Offers detailed Keil operational instructions. Available at <a href="http://www.keil.com/support/man/docs/uv4/">http://www.keil.com/support/man/docs/uv4/</a> .
Bluetooth Core Spec v5.1	Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at <a href="https://www.bluetooth.com/specifications/bluetooth-core-specification/">https://www.bluetooth.com/specifications/bluetooth-core-specification/</a> .
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at <a href="http://www.bluetooth.com/specifications/gatt">www.bluetooth.com/specifications/gatt</a> .
GProgrammer User Manual	Lists GProgrammer operational instructions including downloading firmware to and encrypting firmware on GR551x SoCs.

## 2 Environment Setup

This chapter introduces how to rapidly set up an operating environment for GR551x Fault Trace Module.

### 2.1 Preparation

- **Hardware preparation**

Table 2-1 Hardware preparation

Name	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit <a href="http://www.segger.com/products/debug-probes/j-link/">www.segger.com/products/debug-probes/j-link/</a> .
Development board	GR5515 Starter Kit Board (GR5515 SK Board), the target device on which Fault Trace Module runs
Connection cable	Micro USB 2.0 serial cable
Android phone	Phones running on Android 4.4 (KitKat) and later versions

- **Software preparation**

Table 2-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at <a href="http://www.segger.com/downloads/jlink/">www.segger.com/downloads/jlink/</a> .
Keil MDK	An integrated development environment (IDE). Version 5.20 or later is required for GR551x SDK. Available at <a href="https://www.keil.com/demo/eval/arm.htm">https://www.keil.com/demo/eval/arm.htm</a> .
GProgrammer (Windows)	A GR551x programming tool. Available in SDK_Folder\tools\GProgrammer.
GRUart (Windows)	A GR551x serial port debugging tool. Available in SDK_Folder\tools\GRUart.
GRTtoolbox (Android)	A Bluetooth LE debugging tool for GR551x. Available in SDK_Folder\tools\GRTtoolbox.

 **Note:**

SDK\_Folder is the root directory of GR551x SDK.

### 2.2 Hardware Connection

Connect the GR5515 SK Board to a PC with a Micro USB 2.0 cable.

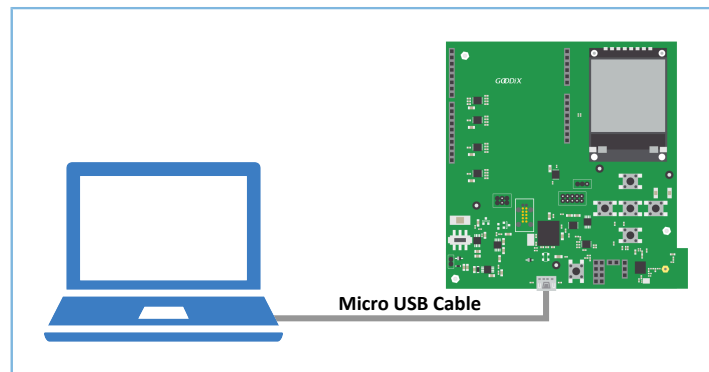


Figure 2-1 Hardware connection

## 2.3 Serial Port Settings

Start GRUart, and configure the serial port parameters in accordance with [Table 2-3](#).

Table 2-3 Configuring serial port parameters on GRUart

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
Select on demand	115200	8	None	1	Uncheck

## 3 Fault Trace Module in Application

This chapter introduces how to add GR551x Fault Trace Module to a project and how to use the Module by taking ble\_app\_hrs as an example.

### 3.1 Import Data to Fault Trace Module to Target Project

Fault Trace Module is optional for running a GR551x-SoC-based project. Before using the Module, add the files of Fault Trace Module to the project directory of ble\_app\_hrs and enable the macro switch of the Module.

#### 3.1.1 Add the Module

1. Open the heart rate example project ble\_app\_hrs.

The source code and project file of the GR551x heart rate example project are in SDK\_Folder\projects\ble\ble\_peripheral\ble\_app\_hrs, and project file is in the Keil\_5 folder.

2. Add the source files of Fault Trace Module to the project directory of ble\_app\_hrs.

Source files of Fault Trace Module are in SDK\_Folder\components\libraries\fault\_trace and SDK\_Folder\components\libraries\app\_error.

Select and right-click gr\_libraries. Choose **Add Existing Files to Group gr\_libraries** to add *fault\_trace.c* and *cortex\_backtrace.c* to gr\_libraries. The directory is then shown as below:

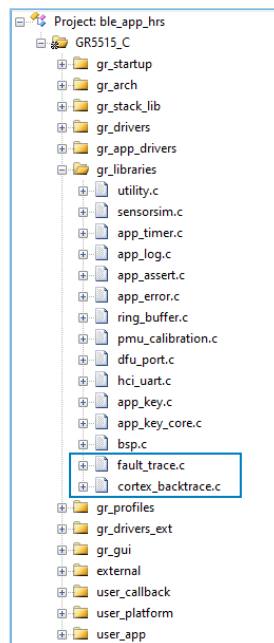


Figure 3-1 Adding source files of Fault Trace Module to the project directory

#### 3.1.2 Enable the Module

Open *user\_app/custom\_config.h* in the directory. Find out the macro `SYS_FAULT_TRACE_ENABLE` of the Module, and set `SYS_FAULT_TRACE_ENABLE` to 1.



### 3.1.3 Initialize the Module

Call `fault_trace_db_init()` in `app_log_assert_init()` in `user_periph_setup.c`, to initialize the Module.

```
static void app_log_assert_init(void)
{
    app_log_init_t log_init;

    log_init.filter.level           = APP_LOG_LVL_DEBUG;
    log_init.fmt_set[APP_LOG_LVL_ERROR] = APP_LOG_FMT_ALL & (~APP_LOG_FMT_TAG);
    log_init.fmt_set[APP_LOG_LVL_WARNING] = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_INFO] = APP_LOG_FMT_LVL;
    log_init.fmt_set[APP_LOG_LVL_DEBUG] = APP_LOG_FMT_LVL;

    app_log_init(&log_init, bsp_uart_send, bsp_uart_flush);

    fault_trace_db_init();
}
```

After adding the Module to the right directory, enabling, and initializing the Module, program the firmware generated based on the compiled program to the GR5515 SK Board by following instructions in *GR551x Developer Guide*.

If a HardFault or an Assert fault occurs on the GR5515 SK Board when running the project, relevant fault trace data will be stored in the NVDS of the GR551x SoC. The data will be kept until you do global erase on the Flash of the SoC.

## 3.2 Read Fault Trace Data

You can read the fault trace data in the NVDS by following any of the three approaches:

1. Open GRToolbox on an Android phone, and get the fault trace data from the NVDS of the GR5515 SK Board via Bluetooth.
2. Read the fault trace data from the NVDS of the GR5515 SK Board by using GProgrammer on a PC.
3. Call the relevant APIs in the project to read the fault trace data.

---

#### Note:

Before you get fault trace data via Bluetooth or by calling the APIs, make sure the Fault Trace Module is added to the firmware of the GR5515 SK Board. This is not required if you choose to get the data through GProgrammer.

---

### 3.2.1 Read Data via Bluetooth

To read data via Bluetooth, make sure the Fault Trace Module is added and the Log Notification Service (LNS) is running on the GR5515 SK Board.

If LNS is not available on the GR5515 SK Board, add LNS to `ble_app_hrs`.

The source file of LNS is in `SDK_Folder\components\profile\lns`.

Select and right-click `gr_profiles` under the project directory of `ble_app_hrs`. Choose **Add Existing Files to Group `gr_profiles`** to add `lns.c` to `gr_profiles`. The directory is then shown as below:

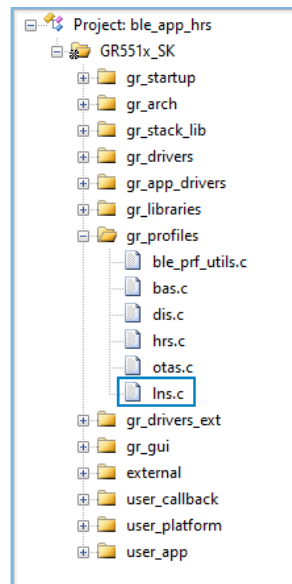


Figure 3-2 Adding the LNS file to the project directory

Call `lns_service_init()` to initialize the LNS.

**Note:**

The code below is available in `SDK_Folder\projects\ble\ble_peripheral\ble_app_hrs\Src\user\user_app.c` in the SDK, and available in `GR5515_SK\user_app\user_app.c` in the example project directory.

```
static void services_init(void)
{
    ...
    lns_service_init(NULL);

    #if DFU_ENABLE
        dfu_service_init(NULL);
    #endif
}
```

**Note:**

The code line in bold is the newly added `services_init()` function to initialize the LNS.

After the project is properly configured, run the firmware generated from the example project on GR5515 SK Board.

1. Open GRToolbox on an Android phone and connect the phone with the GR5515 SK Board. **Log Notification Service** is then displayed, as shown in [Figure 3-3](#).

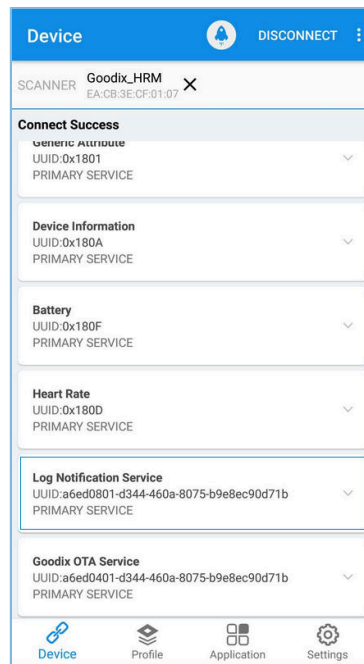



Figure 3-3 Successful discovery of LNS after connecting the phone to the Board on GRTtoolbox

2. Tap  in the upper-right corner, and choose **Read Log**, as shown in Figure 3-4.

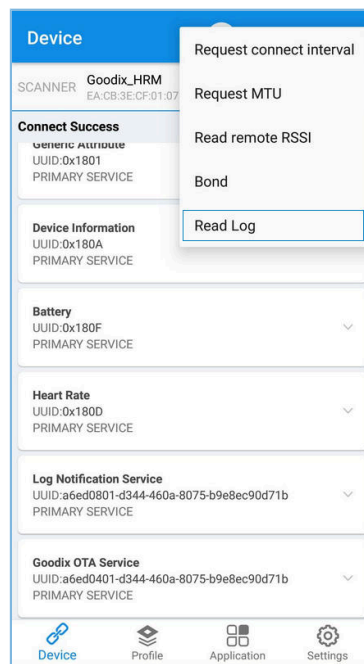


Figure 3-4 To read device logs

3. Tap **READ** in the **Read Log** pop-up box, to read the fault trace data stored on the board.



Figure 3-5 Successfully reading device logs

### 3.2.2 Read Data Through GProgrammer

Connect the PC with the GR5515 SK Board that you wish to read fault trace data from, and start GProgrammer on the PC.


Click  on the left bar of the main user interface of GProgrammer, to enter the **Device Log** page.



Figure 3-6 Device Log on GProgrammer

Click **Read** in the bottom-right corner of the page, to read the fault trace data from the NVDS of the GR5515 Starter Kit Board, as shown in [Figure 3-7](#):



Figure 3-7 Fault trace data displayed on the **Device Log** page

### 3.2.3 Read Data by Calling APIs in the Project

Fault Trace Module provides you with APIs to read the data. To get the fault trace data, you need to call the relevant APIs in the project, and output data through serial ports.

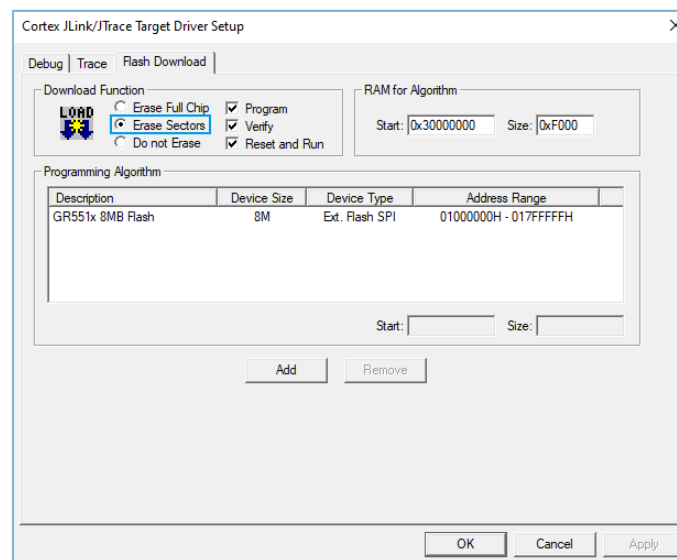


Figure 3-8 Setting the erase type

#### Note:

If the firmware is programmed with the project in Keil MDK, select **Erase Sectors** for erase, as shown in [Figure 3-8](#). Avoid selecting **Erase Full Chip**, which will also erase the fault trace data in the NVDS of the Board.

The code below is an example to read fault trace data by calling APIs in the project.

```

sdk_err_t error_code;
uint8_t fault_trace_data[1000] = {0};
uint32_t data_len = 1000;
error_code = fault_db_records_dump(fault_trace_data, &data_len);
APP_ERROR_CHECK(error_code);
for (uint32_t i = 0; i < data_len; i++)
{
    APP_LOG_RAW_INFO("%c", fault_trace_data[i]);
}

```

}

**Note:**

Make sure the UART module and the APP LOG module have been initialized before reading fault trace data. In the example project, the two modules are initialized by running `app_periph_init()`. For more information, see “Section 4.3.3.1 Modify the main() Function” in *GR551x Developer Guide*.

An example of fault trace data obtained through GRUart is displayed in [Figure 3-9](#).

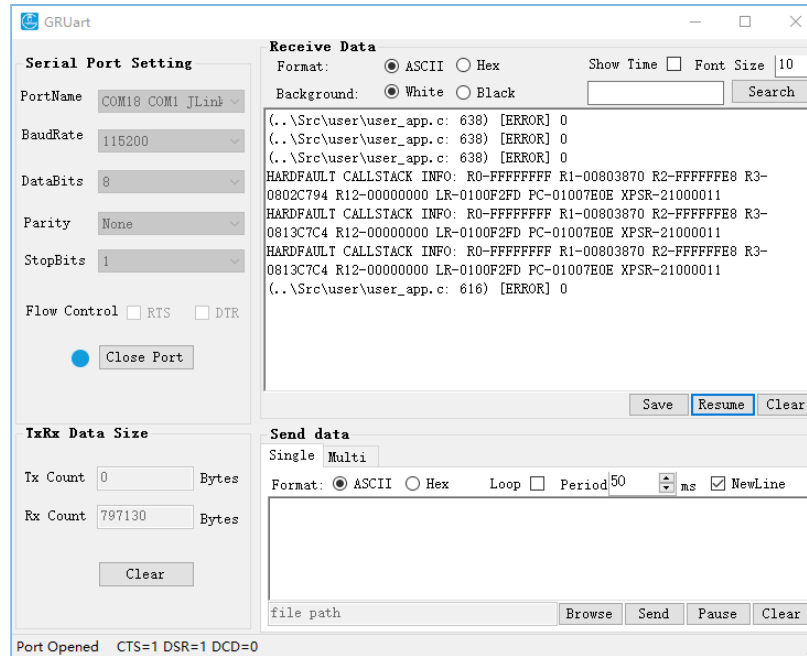


Figure 3-9 Fault trace data displayed on GRUart

### 3.3 Operation Demonstration

This section explains the functionalities of Fault Trace Module by taking the HardFault scenario, a common scenario as an example.

1. Add the code that causes the HardFault to the code of `ble_app_hrs`.

```
static void heartrate_service_process_event(hrs_evt_t *p_hrs_evt)
{
    sdk_err_t error_code;
    switch (p_hrs_evt->evt_type)
    {
        case HRS_EVT_NOTIFICATION_ENABLED:
            error_code = app_timer_start(s_heart_rate_meas_timer_id,
HEART_RATE_MEAS_INTERVAL, NULL);
            APP_ERROR_CHECK(error_code);

            error_code = app_timer_start(s_rr_interval_meas_timer_id, RR_INTERVAL_INTERVAL,
NULL);
            APP_ERROR_CHECK(error_code);
            APP_LOG_DEBUG("Heart Rate Notification Enabled." );

            //Access illegal address
    }
}
```

```

    *(volatile uint32_t*) (0xFFFFFFFF) |= (1 << 0);
    break;
    ...
}
}

```

**Note:**

The code above is available in `SDK_Folder\components\profiles\hrs\hrs.h` in the SDK, and available in `GR5515_SK\gr_profiles\hrs.c` in the example project directory.

The code lines in bold are newly added which explain the cause of the HardFault.

Enable the **Heart Rate** notification and a new code line that leads to access to the invalid address is displayed. A HardFault will occur after you view the **Heart Rate** notification on GRToolbox.

2. Compile the project and generate the firmware. Download the firmware to the GR5515 SK Board. Run the project in debug mode, and set breakpoints on the code line where the HardFault is caused. Connect the phone with the GR5515 SK Board on GRToolbox, and tap **N** on the right of **Heart Rate Measurement** to view **Heart Rate** notification.

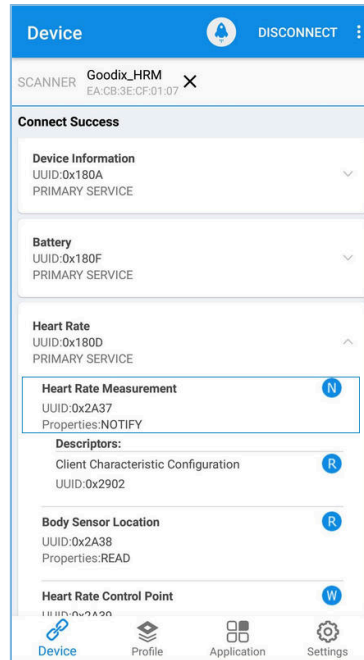


Figure 3-10 To view **Heart Rate** notification

By doing so, the project stops running at the breakpoint. Values of the registers are shown in the left pane, as shown in [Figure 3-11](#).

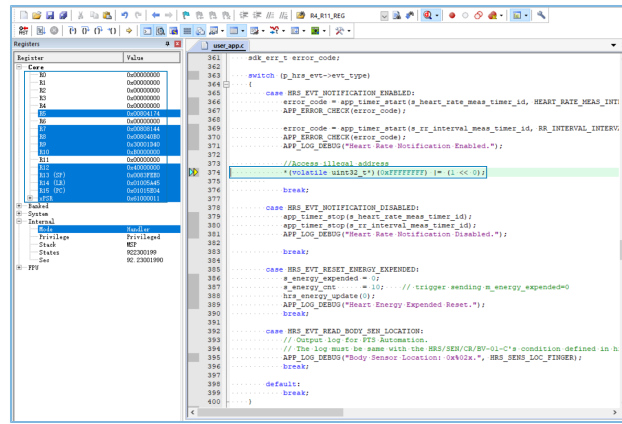


Figure 3-11 Debug interface before the HardFault occurs

3. Press F11 to step through code running. The project runs into the function that causes the HardFault. Fault trace data before the HardFault occurs is stored in the NVDS.

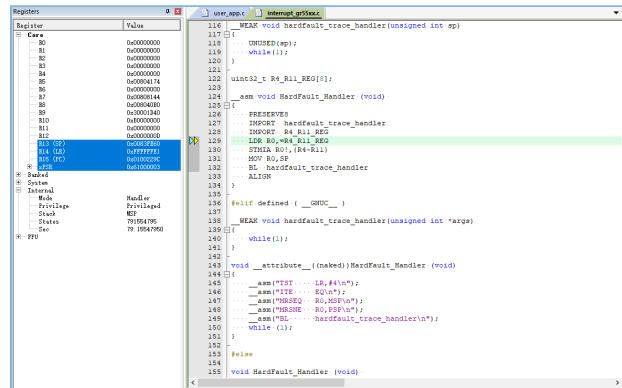


Figure 3-12 Entering the function causing HardFault

4. Reset the GR5515 SK Board after the Board exiting the debug mode. Read the fault trace data from the Board through Bluetooth (by following the instructions in “Section 3.2.1 Read Data via Bluetooth”). The data is displayed as below:



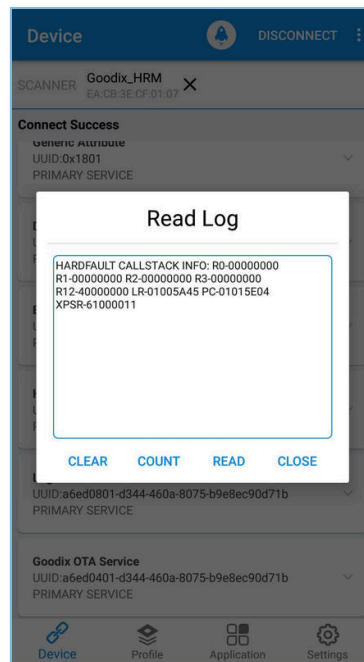


Figure 3-13 Reading fault trace data through Bluetooth

The register values on the debug interface (Figure 3-12) match with that shown in fault trace data (Figure 3-13), which proves that the Fault Trace Module records the fault trace data of the HardFault.

## 4 Module Details

The read and write of Fault Trace Module are enabled by the APIs for NVDS. This chapter elaborates on the mechanisms for tracing HardFaults and Assert faults, and the mechanism for data tracing through Bluetooth connection.

### 4.1 HardFault Data Tracing

When a HardFault occurs, registers PSR, R15 (PC), R14 (LR), R3, R2, R1, and R0, which are controlled by the processor at the hardware level, are pushed onto stack in order, and are included in HardFault\_Handler() for exception handling.

#### Note:

The code below is available in

SDK\_Folder\projects\ble\ble\_peripheral\ble\_app\_hrs\Src\user\user\_app.c in the SDK, and available in GR5515\_SK\gr\_arch\interrupt\_gr55xx.c in the example project directory.

```
uint32_t R4_R11_REG[8];

__asm void HardFault_Handler (void)
{
    PRESERVE8
    IMPORT hardfault_trace_handler
    IMPORT R4_R11_REG
    LDR R0, =R4_R11_REG
    STMIA R0!, {R4-R11}
    MOV R0, SP
    BL hardfault_trace_handler
    ALIGN
}
```

HardFault\_Handler() enables saving the values of R4 to R11 in the global array R4\_R11\_REG when a HardFault occurs. The pointer SP is assigned to R0, and SP will be a parameter in hardfault\_trace\_handler(), the function that will be called in the next step.

#### Note:

The code below is available in

SDK\_Folder\app\components\libraries\fault\_trace\fault\_trace.c in the SDK, and available in GR5515\_SK\gr\_libraries\fault\_trace.c in the example project directory.

```
void hardfault_trace_handler(unsigned int sp)
{
    unsigned int stacked_r0;
    unsigned int stacked_r1;
    unsigned int stacked_r2;
    unsigned int stacked_r3;
    unsigned int stacked_r12;
    unsigned int stacked_lr;
    unsigned int stacked_pc;
    unsigned int stacked_psr;

    stacked_r0 = ((unsigned long *)sp)[0];
    stacked_r1 = ((unsigned long *)sp)[1];
```

```

stacked_r2 = ((unsigned long *)sp)[2];
stacked_r3 = ((unsigned long *)sp)[3];
stacked_r12 = ((unsigned long *)sp)[4];
stacked_lr = ((unsigned long *)sp)[5];
stacked_pc = ((unsigned long *)sp)[6];
stacked_psr = ((unsigned long *)sp)[7];

memset(s_fault_info, 0, FAULT_INFO_LEN_MAX);

sprintf(s_fault_info,
        "HARDFFAULT CALLSTACK INFO: R0-%08X R1-%08X R2-%08X R3-%08X R12-%08X LR-%08X
        PC-%08X XPSR-%08X\r\n",
        stacked_r0, stacked_r1, stacked_r2, stacked_r3, stacked_r12, stacked_lr,
        stacked_pc, stacked_psr);

fault_db_record_add((uint8_t *)s_fault_info, strlen(s_fault_info));

while(1);
}

```

The parameter `sp` enables `hardfault_trace_handler()` to read the register values from the stack, and write the values to `s_fault_info`. `fault_db_record_add()` is then called to write the values to NVDS.

The fault trace data format of a HardFault is shown below:

```

HARDFFAULT CALLSTACK INFO: R0-FFFFFFFF R1-00803870 R2-FFFFFFE8 R3-0802C794 R12-00000000
LR-0100F2FD PC-01007E0E XPSR-21000011}

```

The fault trace data above corresponds to the values of registers: R0, R1, R2, R3, R12, R14 (LR), R15 (PC), and PSR (XPSR) when a HardFault occurs.

#### Note:

`HardFault_Handler()` enables saving the values of registers from R4 to R11 in the global array `R4_R11_REG` when a HardFault occurs. You can use `hardfault_trace_handler()` on demand, and write the obtained values into NVDS.

## 4.2 Assert Fault Data Tracing

The Assert method is used to debug software by identifying errors in code. The Assert module is available in `SDK_Folder\components\libraries\app_assert` in the SDK.

#### Note:

The code below is available in `SDK_Folder\components\libraries\ap_assert\app_assert.h` in the SDK, and available in `GR5515_SK\gr_libraries\app_assert.h` in the example project directory.

```

#define APP_ASSERT_CHECK(EXPR)
do
{
    if (! (EXPR))
    {
        app_assert_handler(#EXPR, __FILE__, __LINE__);
    }
} while(0)

```

When `APP_ASSERT_CHECK(EXPR)` is called and `EXPR = 0`, `app_assert_handler()` will be called.

**Note:**

The code below is available in SDK\_Folder\components\libraries\ap\_assert\app\_assert.c in the SDK, and available in GR5515\_SK\gr\_libraries\app\_assert.c in the example project directory.

```
void app_assert_handler(const char *expr, const char *file, int line)
{
    if (s_assert_cbs.assert_err_cb)
    {
        s_assert_cbs.assert_err_cb(expr, file, line);
    }
}
```

Callback functions will be called in the handler, to output fault trace data through serial ports.

```
/**@brief Assert callbacks.*/
typedef struct
{
    assert_err_cb_t   assert_err_cb;    /**< Assert error type callback. */
    assert_param_cb_t assert_param_cb;  /**< Assert parameter error type callback. */
    assert_warn_cb_t  assert_warn_cb;   /**< Assert warning type callback. */
}sys_assert_cb_t;
```

Three callbacks are called in the Assert module, with each corresponding to a different Assert parameter format and fault trace data (see the source code of app\_assert in *app\_assert.c*). By default, app\_assert\_handler() calls assert\_err\_cb(). You can use app\_assert\_handler() on demand, and call other callback functions.

Callback functions of the Assert module can output fault trace data through serial ports. Fault Trace Module helps the three callbacks of the Assert module cover previous implementation (the three callbacks were implemented as weak functions) and save fault trace data in NVDS. assert\_err\_cb() is implemented as below:

**Note:**

The code below is available in SDK\_Folder\components\libraries\fault\_trace\fault\_trace.c in the SDK, and available in GR5515\_SK\gr\_libraries\fault\_trace.c in the example project directory.

```
static void assert_err_cb(const char *expr, const char *file, int line)
{
    __disable_irq();

    uint32_t      expre_len      = 0;
    uint32_t      file_name_len  = 0;

    file_name_len=(ASSERT_FILE_NAME_LEN < strlen(file)) ?
ASSERT_FILE_NAME_LEN:strlen(file);
    expre_len = (ASSERT_EXPR_NAME_LEN < strlen(expr)) ?
ASSERT_EXPR_NAME_LEN : strlen(expr);

    memset(&s_assert_info, 0, sizeof(assert_info_t));
    memcpy(s_assert_info.file_name, file, file_name_len);
    memcpy(s_assert_info.expr, expr, expre_len);

    s_assert_info.assert_type = ASSERT_ERROR;
    s_assert_info.file_line   = line;

    assert_info_save(&s_assert_info);
    while(1);
}
```

```
}

```

You can save actual parameter names (param), function names and paths for calling APP\_ASSERT\_CHECK(), and number of code lines to the structure, by using assert\_err\_cb(), and save the data to NVDS in designated format by calling the APIs for NVDS.

An example of the fault trace data of Assert faults is shown below:

```
(..\Src\user\user_app.c: 638) [ERROR] param

```

The fault trace data shows the path for the Assert fault (..\Src\user\user\_app.c: 638), fault type (ERROR), and name of the actual parameter (param).

### 4.3 Data Tracing Through Bluetooth

You can control the Fault Trace Module on the GR5515 Starter Kit Board through Bluetooth, which is enabled by LNS. LNS provides specific characteristics to receive control commands and send data.

LNS characteristics include Log Information and Log Control Point, with details listed in [Table 4-1](#).

Table 4-1 LNS Characteristics

Characteristic	UUID	Type	Support	Security	Property
Log Information	A6ED0802-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Notify
Log Control Point	A6ED0803-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write, Indicate

- Log Information: used to send fault trace data (Notify)
- Log Control Point: used to receive commands (Write) and return information (Indicate)

This section elaborates on the mechanism that Bluetooth controls Fault Trace Module by introducing the implementation of LNS.

#### Note:

The code below is available in SDK\_Folder\app\components\libraries\ap\_assert\ap\_assert.c in the SDK, and available in GR5515\_SK\gr\_libraries\ap\_assert.c in the example project directory.

```
static void lns_write_att_cb(uint8_t conn_idx,
                           const gatts_write_req_cb_t *p_param)
{
    ...
    switch (tab_index)
    {
        ...
        case LNS_IDX_LOG_CTRL_PT_VAL:
        {
            switch (p_param->value[0])
            {
                case LNS_CTRL_PT_TRACE_STATUS_GET:
                    event.evt_type = LNS_EVT_TRACE_STATUS_GET;
                    break;

                case LNS_CTRL_PT_TRACE_INFO_DUMP:
                    event.evt_type = LNS_EVT_TRACE_INFO_DUMP;
            }
        }
    }
}

```

```

        break;

        case LNS_CTRL_PT_TRACE_INFO_CLEAR:
            event.evt_type = LNS_EVT_TRACE_INFO_CLEAR;
            break;

        default:
            break;
    }
}
...
if (BLE_ATT_ERR_INVALID_HANDLE != cfm.status && LNS_EVT_INVALID != event.evt_type)
{
    lns_evt_handler(&event);
}
}

```

`lns_write_att_cb()` is a callback function written to LNS. The Client writes `LNS_CTRL_PT_TRACE_STATUS_GET (0x01)`, `LNS_CTRL_PT_TRACE_INFO_DUMP (0x02)`, and `LNS_CTRL_PT_TRACE_INFO_CLEAR (0x03)` to Log Control Point. Any one of the three can trigger a type of event (`event.evt_type`), and calls the registered event handling function `lns_evt_handler()`.

#### Note:

The code below is available in `SDK_Folder\components\libraries\ap_assert\ap_assert.c` in the SDK, and available in `GR5515_SK\gr_libraries\ap_assert.c` in the example project directory.

```

static void lns_evt_handler(lns_evt_t *p_evt)
{
    uint8_t trace_log_num = 0;

    switch (p_evt->evt_type)
    {
        case LNS_EVT_TRACE_STATUS_GET:
            trace_log_num = fault_db_records_num_get();
            lns_log_status_send(p_evt->conn_idx, trace_log_num);
            break;

        case LNS_EVT_TRACE_INFO_DUMP:
            lns_log_info_send(p_evt->conn_idx);
            break;

        case LNS_EVT_TRACE_INFO_CLEAR:
            fault_db_record_clear();
            break;
    }

    if (LNS_EVT_INVALID != p_evt->evt_type && s_lns_env.evt_handler)
    {
        s_lns_env.evt_handler(p_evt);
    }
}

```

The corresponding function in the event handling function of LNS is called for each type of event. Each value written to Log Control Point by the Client is associated with an event type.

- When the Client writes 0x01, `fault_db_records_num_get()` and `Ins_log_status_send()` are called, to read the number of fault trace data entries and send the number to the peer device.
- When the Client writes 0x02, `Ins_log_info_send()` is called, to read the fault trace data and send the data to the peer device.
- When the Client writes 0x03, `fault_db_record_clear()` is called, to clear the fault trace data.

As shown in [Figure 3-5](#), **COUNT**, **READ**, and **CLEAR** on the GRToolbox interface are implemented by writing 0x01, 0x02, and 0x03 to Log Control Point of the Slave LNS. You can also implement these functionalities by enabling notifications of Log Information and Log Control Point, and writing the corresponding value.

## 5 FAQ

This chapter describes the possible problems when using Fault Trace Module, analyzes the causes, and provides solutions.

### 5.1 Why do I Fail to Read Fault Trace Data on GProgrammer?

- Description

Why do I fail to read fault trace data on GProgrammer, and no data for **USER Parameters** is obtained?

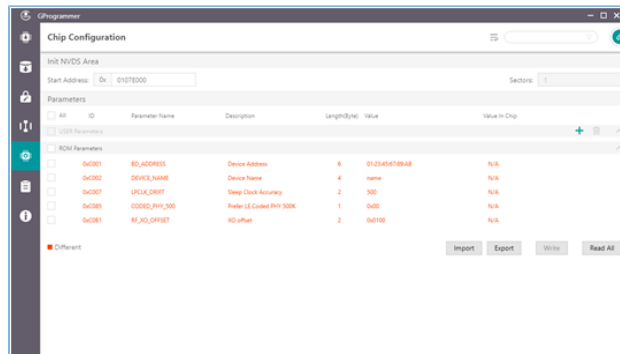


Figure 5-1 Failing to obtain **USER Parameters**

- Analysis

This may be because the start address on the interface has not been reset. The start address of NVDS shall be reset every time when GProgrammer is started.

- Solution

Enter "010FF000" in the **Start Address** field. If the NVDS is reallocated, set the start address accordingly.

### 5.2 Why do I Fail to Read Fault Trace Data by Calling APIs in the Project?

- Description

Why do I fail to read fault trace data by calling APIs in the project, and obtain no data output on GRUart? If I check the returned value from `fault_db_records_dump()` by using `APP_ERROR_CHECK()`, GRUart shows information as follows:



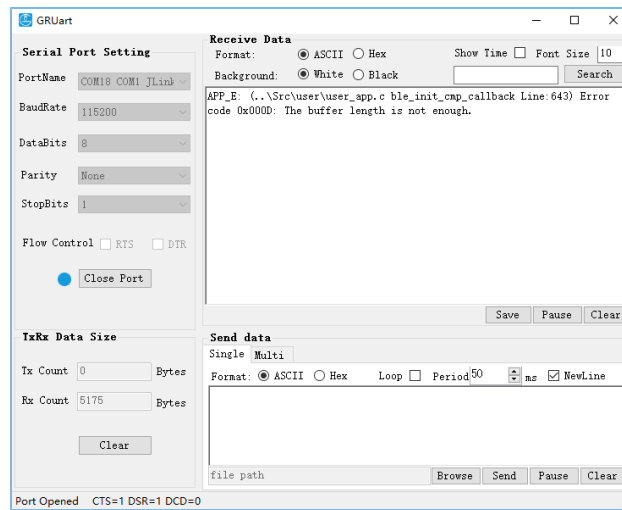


Figure 5-2 Serial port output when checking returned value in case of data read failure

- **Analysis**  
The buffer size for storing fault trace data is insufficient.
- **Solution**  
Increase the buffer size for storing fault trace data.