GR551x Peripheral Examples Application Manual

Version: 1.8

Release Date: 2021-04-25

Shenzhen Goodix Technology Co., Ltd.

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

Preface

Purpose

This document introduces how to use and modify peripheral examples in a GR551x SDK, to help users quickly get started with secondary development.

Audience

This document is intended for:

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

Release Notes

This document is the sixth release of *GR551x Peripheral Examples Application Manual*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Deleted SysTick-related projects, and updated DMA instance to DMA channel.
1.5	2020-05-30	Added formulas in “FAQ about ADC”.
1.6	2020-06-30	• Corrected the firmware name in “Test and Verification”. • Updated the reference voltage for ADC in “ADC Battery” and “ADC Tempereature”. • Revised description on the processes for example projects in “HAMC” and “PKC”.
1.7	2020-12-18	• Added descriptions on PWM alignment modes in “PWM”. • Added limitations on using RNG_OUTPUT_FR0_S0 in “RNG”.
1.8	2021-04-25	Deleted DMA description in “AES”.

Contents

1 Introduction

This document introduces how to use and modify peripheral examples in a GR551x SDK, which allows users to quickly get started with secondary development.

You can refer to the following documents before using and modifying a peripheral example.

Table 1-1 Reference documents

Name	Description
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
GR551x Datasheet	Provides overview, pinout, memory, PMU, clock, peripherals, security cores, communications subsystems, and packaging information of GR551x SoCs.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at http://www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/.

2 Initial Operation

This chapter introduces how to quickly verify a peripheral example in a GR551x SDK.

Note:

SDK_Folder is the root directory of GR551x SDK.

2.1 Preparation

Perform the following tasks before verifying and testing a peripheral example.

- **Hardware preparation**

Table 2-1 Hardware preparation

Name	Description
Development board	GR5515 Starter Kit Board (GR5515 SK Board). Two boards are required for some peripheral examples.

- **Software preparation**

Table 2-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
GRUart (Windows)	A GR551x serial port debugging tool. Available in SDK_`Folder\tools\GRUart`.

2.2 Hardware Connection

Connect a GR5515 SK Board to a PC with a Micro USB 2.0 cable, as shown in the figure below.

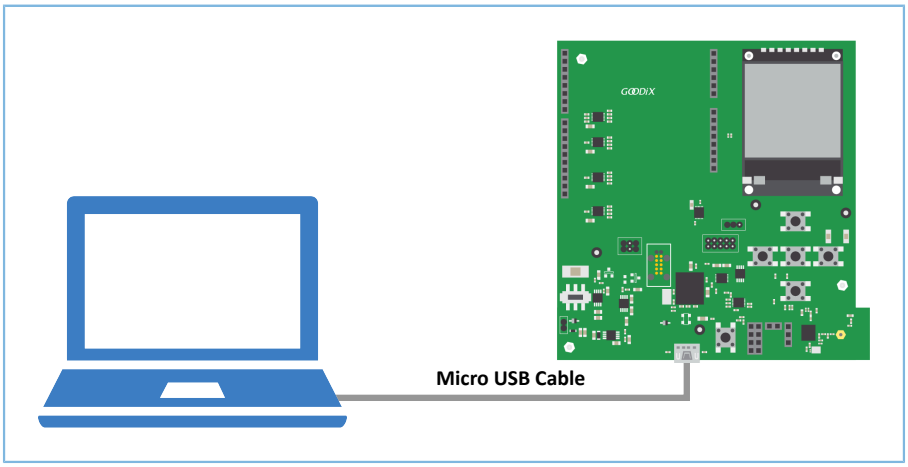

Figure 2-1 Hardware connection

2.3 Firmware Download

Take GPIO LED as an example. Download *gpio_led_sk_r2_fw.bin* firmware of the example to a GR5515 SK Board. For details, refer to *GProgrammer User Manual*.

Note:

The *gpio_led_sk_r2_fw.bin* file is in `SDK_Folder\projects\peripheral\gpio\gpio_led\build\`.

2.4 Serial Port Settings

Start GRUart, and configure the serial ports according to the parameters in the table below.

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 Peripheral Example Overview

All peripheral examples in a GR551x SDK are stored in SDK_Folder\projects\peripheral\.

Take Analog-to-Digital Converter (ADC) as an example. Double-click the project file adc.uvprojx, to check the project directory structure of the example in Keil, as shown in Figure 3-1 below.

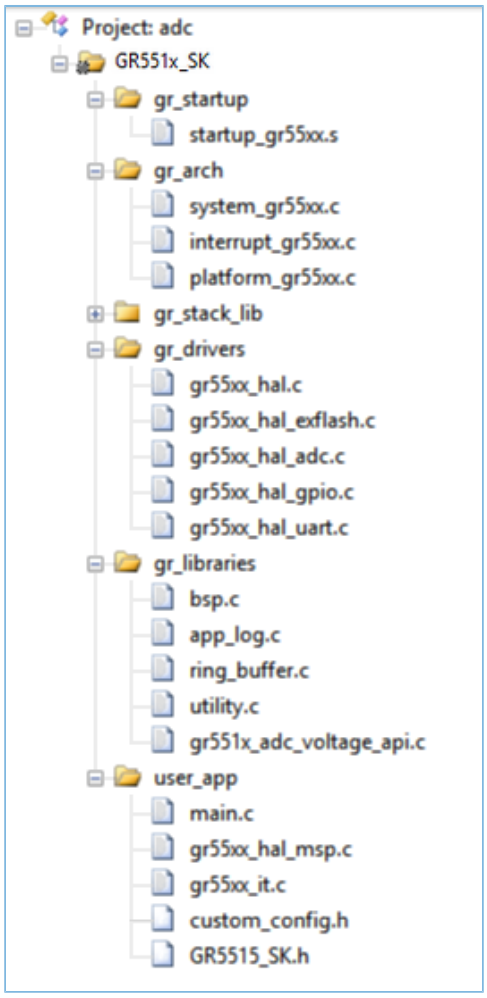

Figure 3-1 adc.uvprojx project structure directory

Table 3-1 lists some files that are common to all peripherals.

Table 3-1 Common peripheral files

Group	File	Description
user_app	main.c	This file contains the main() function that holds initialization and execution code for a peripheral.
	gr55xx_hal_msp.c	This file helps initialize a peripheral in terms of its GPIOs, interrupts, and DMA configurations.
	gr55xx_it.c	This file provides an interrupt entry function for a peripheral and its associated peripherals.
	GR5515_SK.h	This file contains macro defines for GPIO and other common hardware in a peripheral.

- gr55xx_hal_msp.c

  This file initializes and deinitializes GPIOs, interrupts, and other hardware-related components in an ADC peripheral. The ADC example has two such functions:

  hal_adc_msp_init(): initialization function for GPIOs and interrupts. This function calls:

  1. NVIC_ClearPendingIRQ() and hal_nvic_enable_irq() to clear pending DMA interrupts and enable DMA interrupts respectively.
  2. hal_msio_init() to initialize the ADC_P_INPUT_PIN(MSIO_PIN_0) and ADC_N_INPUT_PIN(MSIO_PIN_1) pins as analog input pins.
  3. hal_dma_ini() to initialize the DMA module.

  The code snippet is as follows:

```
void hal_adc_msp_init(adc_handle_t *hadc)
{
    msio_init_t msio_config = MSIO_DEFAULT_CONFIG;

    NVIC_ClearPendingIRQ(DMA_IRQn);
    hal_nvic_enable_irq(DMA_IRQn);

    /* Config input GPIO */
    msio_config.pin  = ADC_P_INPUT_PIN | ADC_N_INPUT_PIN;
    msio_config.mode = MSIO_MODE_ANALOG;
    hal_msio_init(&msio_config);

    /* Configure the DMA handler for Transmission process */
    hadc->p_dma = &s_dma_handle;
    s_dma_handle.p_parent = hadc;
    hadc->p_dma->channel            = DMA_Channel0;
    hadc->p_dma->init.src_request   = DMA_REQUEST_SNSADC;
    hadc->p_dma->init.direction     = DMA_PERIPH_TO_MEMORY;
    hadc->p_dma->init.src_increment = DMA_SRC_NO_CHANGE;
    hadc->p_dma->init.dst_increment = DMA_DST_INCREMENT;
    hadc->p_dma->init.src_data_alignment = DMA_SDATAALIGN_WORD;
    hadc->p_dma->init.dst_data_alignment = DMA_DDATAALIGN_WORD;
    hadc->p_dma->init.mode          = DMA_NORMAL;
    hadc->p_dma->init.priority      = DMA_PRIORITY_LOW;

    hal_dma_init(hadc->p_dma);
}
```

  hal_adc_msp_deinit(): deinitialization function for GPIOs and interrupts. This function calls:

  1. hal_msio_deinit() to restore the ADC_P_INPUT_PIN(MSIO_PIN_0) and ADC_N_INPUT_PIN(MSIO_PIN_1) pins as GPIO pins.
  2. hal_dma_deinit() to deinitialize the DMA module.

  The code snippet is as follows:

```
void hal_adc_msp_deinit(adc_handle_t *hadc)
{
    hal_msio_deinit(ADC_P_INPUT_PIN | ADC_N_INPUT_PIN);
    hal_dma_deinit(hadc->p_dma);
}
```

- gr55xx_it.c

  This file provides an interrupt entry function for the ADC peripheral and its associated modules.

  DMA_IRQHandler(): interrupt handler function for DMA. The hal_adc_conv_cplt_callback() callback function returns the operating results and state of DMA_IRQHandler(). Users can redefine this callback function on demand.

  The code snippet is as follows:

```
void DMA_IRQHandler(void)
{
    hal_dma_irq_handler(g_adc_handle.p_dma);
}
__WEAK void hal_adc_conv_cplt_callback(adc_handle_t *p_adc)
{
}
```

3.1 ADC

The ADC converts continuously changing analog signals into discrete digital signals, making it ideal for detecting the voltages and temperatures of peripherals.

This chapter focuses on using an ADC to measure MSIO input signals, battery voltage, and internal SoC temperature.

3.1.1 ADC

The ADC example in a GR551x SDK adopts two sampling modes: single-ended sampling and differential sampling.

- In single-ended sampling mode, users need to connect a valid signal to Pin MSIO1 with an input range from 0 to (2 x Vref) but no higher than VBAT.
- In differential sampling mode, users need to connect differential signals to Pins MSIO0 and MSIO1 with an input range from –(2 x Vref) to +(2 x Vref). The longitudinal voltage is higher than 0.8 V.

The source code and project file of the ADC example are in SDK_Folder\projects\peripheral\adc\adc, and the project file is in the Keil_5 folder.

3.1.1.1 Code Interpretation

Figure 3-2 shows the workflow of an ADC example project.

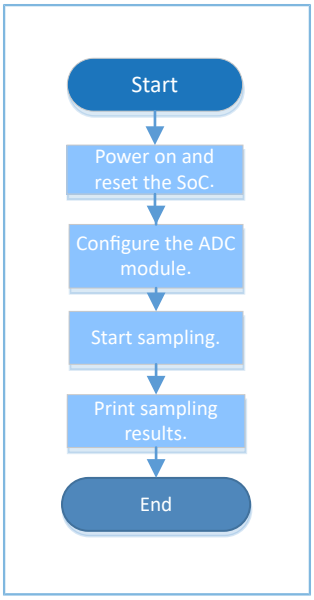

Figure 3-2 ADC example workflow

1. Configure the ADC module.

```
g_adc_handle.init.channel_p  = ADC_INPUT_SRC_IO0;
g_adc_handle.init.channel_n  = ADC_INPUT_SRC_IO1;
g_adc_handle.init.input_mode = ADC_INPUT_SINGLE;
g_adc_handle.init.ref_source = ADC_REF_SRC_BUF_INT;
g_adc_handle.init.ref_value  = ADC_REF_VALUE_1P6;
g_adc_handle.init.clock      = ADC_CLK_1P6M;
```

- init.channel_p and init.channel_n: These parameters set Channels P and N. Input range: ADC_INPUT_SRC_IO0 to ADC_INPUT_SRC_IO4, ADC_INPUT_SRC_TMP, and ADC_INPUT_SRC_BAT. If ADC_INPUT_SRC_IO0 and ADC_INPUT_SRC_IO1 are selected, the Channels P and N are mapped to Pins MSIO0 and MSIO1. Users can modify the mapping to other MSIO pins on demand.
- init.input_mode: This parameter sets the input mode. Options: ADC_INPUT_SINGLE and ADC_INPUT_DIFFERENTIAL. If ADC_INPUT_SINGLE is selected, only sampling on Channel N is allowed. Users can modify the input mode into differential sampling mode on demand.
- init.ref_source: This parameter sets the reference source. Options: ADC_REF_SRC_BUF_INT and ADC_REF_SRC_IO0 to ADC_REF_SRC_IO3. Setting init.ref_source to ADC_REF_SRC_BUF_INT means internal reference source is selected.
- init.ref_value: This parameter sets the voltage of an internal reference source. Options: ADC_REF_VALUE_0P8, ADC_REF_VALUE_1P2, and ADC_REF_VALUE_1P6. Setting init.ref_value to ADC_REF_VALUE_1P6 means the internal reference voltage is 1.6 V with a range from 0 to 3.2 V.
- init.clock: This parameter sets the ADC clock. Options: ADC_CLK_16M, ADC_CLK_1P6M, ADC_CLK_8M, ADC_CLK_4M, ADC_CLK_2M, and ADC_CLK_1M. Setting init.clock to ADC_CLK_1P6M means the clock frequency is 1.6 MHz. Users can modify the setting on demand.

2. The ADC module listed in this chapter reads data in DMA mode during ADC conversion.

(1). Call hal_adc_start_dma() to enable ADC sampling in DMA mode.

The sampling results are read to memories through DMA channels. The code snippet is as follows:

```
hal_adc_start_dma(&g_adc_handle, conversion, TEST_CONV_LENGTH);
```

The code is executed in non-blocking mode, so data reads happen after the ADC status restores to READY. In this example project, the ADC sampling results are returned by the hal_adc_conv_cplt_callback() API. Uses can customize executable operations in this callback function.

The code snippet is as follows:

```
void hal_adc_conv_cplt_callback(adc_handle_t *hadc)
{
    printf("DMA conversion is done.\r\n");
}
```

(2). Call hal_gr551x_adc_voltage_intern() to convert the returned ADC data to a voltage value. The code snippet is as follows:

```
hal_gr551x_adc_voltage_intern(&g_adc_handle, conversion, voltage, TEST_CONV_LENGTH);
```

(3). In scenarios where an external device is used as a reference source, call the following API to convert the returned ADC data to a voltage value:

```
void hal_gr551x_adc_voltage_extern(adc_handle_t *hadc, double vref, uint16_t *inbuf,
                                   double *outbuf, uint32_t buflen)
```

3.1.1.2 Test and Verification

1. Download *adc_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. The GRUart displays the boot info and other logs of the ADC module.
4. You can view the calculated ADC voltage in the **Receive Data** pane on GRUart.

3.1.2 ADC DMA UART

The ADC DMA UART example directly transmits ADC sampling data to PCs or other peripherals using UART interfaces through DMA. This method facilitates high-speed data transfer without relying on CPUs.

The source code and project file of the ADC DMA UART example are in `SDK_Folder\projects\peripheral\adc\adc_dma_uart`, and the project file is in the Keil_5 folder.

3.1.2.1 Code Interpretation

Figure 3-3 shows the workflow of an ADC DMA UART example project.

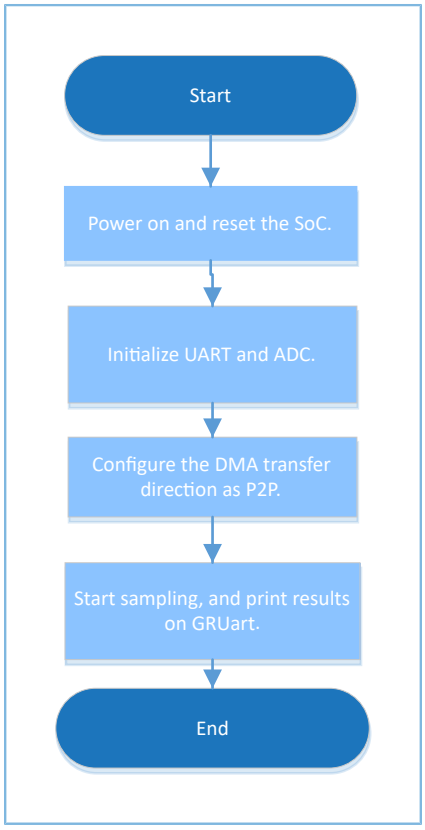

Figure 3-3 ADC DMA UART example workflow

1. Configure the ADC module.

```
g_adc_handle.init.channel_p  = ADC_INPUT_SRC_IO0;
g_adc_handle.init.channel_n  = ADC_INPUT_SRC_IO1;
g_adc_handle.init.input_mode = ADC_INPUT_SINGLE;
g_adc_handle.init.ref_source = ADC_REF_SRC_BUF_INT;
g_adc_handle.init.ref_value  = ADC_REF_VALUE_1P6;
g_adc_handle.init.clock = ADC_CLK_1P6M;
```

For more information about ADC configurations, see “Section 3.1.1 ADC”.

2. Configure the UART TX threshold and DMA transfer unit.

```
ll_adc_set_thresh(16);
ll_uart_set_tx_fifo_threshold(uart_handle.p_instance, LL_UART_TX_FIFO_TH_CHAR_2);
ll_dma_set_source_burst_length(DMA, hadc->p_dma->channel, LL_DMA_SRC_BURST_LENGTH_8);
ll_dma_set_destination_burst_length(DMA, hadc->p_dma->channel, LL_DMA_DST_BURST_LENGTH_8);
```

- Set the ADC threshold to 16, which means a DMA transfer request is made when the ADC takes 16 data samples.
- Set the UART TX threshold to 2, which means a DMA transfer request is made when the UART TX FIFO contains less than two data entries.
- Set the DMA burst length to 8, which means eight data entries are transferred at a time to reduce DMA read times and improve DMA efficiency.

3. Sample data.

```
__HAL_ADC_ENABLE_CLOCK(hadc);
hal_dma_start(hadc->p_dma, (uint32_t)&MCU_SUB->SENSE_ADC_FIFO,
              (uint32_t)&UART1->RBR_DLL_THR, TEST_CONV_LENGTH >> 1);
hal_dma_poll_for_transfer(hadc->p_dma, 1000);
__HAL_ADC_DISABLE_CLOCK(hadc);
```

- Enable ADC clock.
- Call hal_dma_start() to start DMA transfer.
- Poll for hal_dma_poll_for_transfer() to wait for completion of the DMA transfer. The DMA transfer results are transmitted to GRUart through UART1.

3.1.2.2 Test and Verification

1. Download *adc_dma_uart_sk_r2_fw.bin* to two GR5515 SK Boards through GProgrammer.
2. Connect the serial ports of the boards, UART0 and UART1, to the PC, start GRUart, and configure the serial ports on the GRUart as described in "Section 2.4 Serial Port Settings". Two sets of GRUart are required for this example.
3. The GRUart connecting to UART0 prints the debugging information of the boards, and the GRUart connecting to UART1 prints ADC sampling data in hexadecimal system.

3.1.3 ADC Battery

The ADC Battery example enables the ADC to measure battery voltages.

The source code and project file of the ADC Battery example are in SDK_`Folder\projects\peripheral\adc\battery`, and the project file is in the Keil_5 folder.

3.1.3.1 Code Interpretation

Figure 3-4 shows the workflow of an ADC Battery example project.

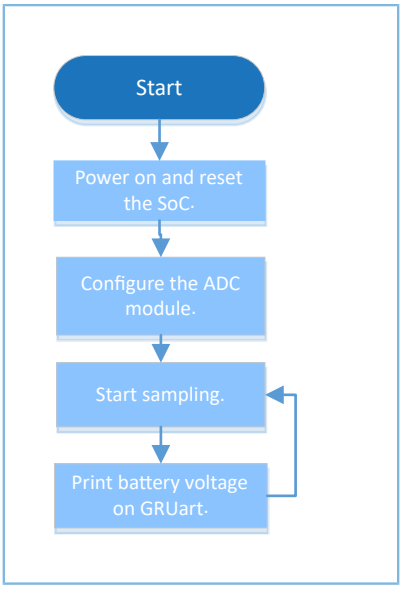

Figure 3-4 ADC Battery example workflow

1. Configure the ADC module.

```
g_adc_handle.init.channel_p = ADC_INPUT_SRC_BAT;
g_adc_handle.init.channel_n = ADC_INPUT_SRC_BAT;
g_adc_handle.init.input_mode = ADC_INPUT_SINGLE;
g_adc_handle.init.ref_source = ADC_REF_SRC_BUF_INT;
g_adc_handle.init.ref_value = ADC_REF_VALUE_0P8;
g_adc_handle.init.clock = ADC_CLK_1P6M;
```

   For more information about ADC configurations, see "Section 3.1.1 ADC". In this example, set both init.channel_p and init.channel_n to ADC_INPUT_SRC_BAT; set init.ref_value to ADC_REF_VALUE_0P8, which means the reference voltage is 0.85 V.

Note:

A measurable VBAT ranges from 2.0 V to 3.8 V. When setting an ADC channel to ADC_INPUT_SRC_BAT, the signals from a voltage divider (input range: 0 to 1.6 V) are measured. Therefore, the reference voltage should be set to 0.85 V to improve measurement accuracy.

---

2. Call hal_gr551x_vbat_init() to initialize the ADC Battery module.
3. Call hal_gr551x_vbat_read() to read the battery voltage. The API returns the current battery voltage (unit: V).

3.1.3.2 Test and Verification

1. Download *battery_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the converted battery voltage in the **Receive Data** pane on GRUart.

3.1.4 ADC Temperature

The ADC Temperature example enables an ADC to measure the internal temperature of an SoC.

The source code and project file of the ADC Temperature example are in `SDK_Folder\projects\peripheral\adc\temperature`, and the project file is in the Keil_5 folder.

3.1.4.1 Code Interpretation

Figure 3-5 shows the workflow of an ADC Temperature example project.

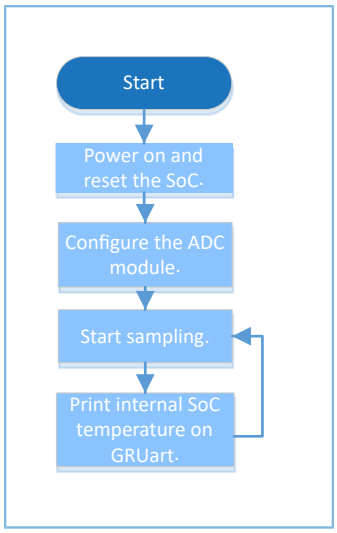

Figure 3-5 ADC Temperature example workflow

1. Configure the ADC module.

```
g_adc_handle.init.channel_p = ADC_INPUT_SRC_TMP;
g_adc_handle.init.channel_n = ADC_INPUT_SRC_TMP;
g_adc_handle.init.input_mode = ADC_INPUT_SINGLE;
g_adc_handle.init.ref_source = ADC_REF_SRC_BUF_INT;
g_adc_handle.init.ref_value = ADC_REF_VALUE_0P8;
g_adc_handle.init.clock = ADC_CLK_1P6M;
```

   For more information about ADC configurations, see "Section 3.1.1 ADC". In this example, set both init.channel_p and init.channel_n to ADC_INPUT_SRC_TMP; set init.ref_value to ADC_REF_VALUE_0P8, which means the reference voltage is 0.85 V.

2. Call hal_gr551x_temp_init() to initialize the ADC Temperature module.
3. Call hal_gr551x_temp_read() to read the temperature. The API returns the internal temperature of an SoC (unit: °C).

3.1.4.2 Test and Verification

1. Download *temp_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".

3. You can view the converted temperature in the **Receive Data** pane on GRUart.

3.2 AES

The Advanced Encryption Standard (AES), known as Rijndael, is one of the most popular symmetric block ciphers after years of development. The AES is applicable for encrypting and decrypting files.

3.2.1 AES

The AES example in a GR551x SDK adopts two encryption modes: Electronic Codebook (ECB) and Cipher Block Chaining (CBC). The example supports three key sizes: 128 bits, 192 bits, and 256 bits.

The AES example outputs both plaintext and ciphertext. A plaintext can be encrypted into different ciphertexts by using different sizes of keys.

- In encryption, the AES module uses a plaintext as input to generate a ciphertext that is compared with original ciphertext to verify the correctness of the AES encryption.
- In decryption, the AES module uses a ciphertext as input to generate a plaintext that is compared with original plaintext to verify the correctness of the AES decryption.

This section focuses on how the AES example encrypts and decrypts data in interrupt and query modes.

The source code and project file of the AES example are in `SDK_Folder\projects\peripheral\aes\aes`, and the project file is in the Keil_5 folder.

3.2.1.1 Code Interpretation

Figure 3-6 shows the workflow of an AES example project.

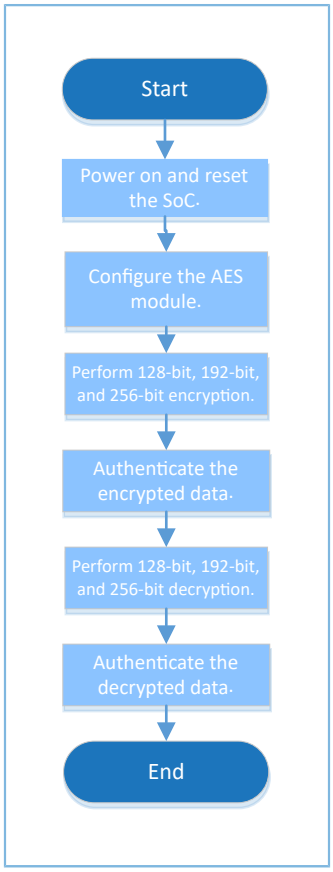

Figure 3-6 AES example workflow

1. Configure the AES module.

   A code snippet of the AES module in ECB mode is as follows:

```
g_aes_handle.p_instance = AES;
g_aes_handle.init.key_size = AES_KEYSIZE_128BITS;
g_aes_handle.init.p_key = (uint32_t *)g_key128_ecb;
g_aes_handle.init.chaining_mode = AES_CHAININGMODE_ECB;
g_aes_handle.init.p_init_vector = NULL;
g_aes_handle.init.dpa_mode = DISABLE;
g_aes_handle.init.p_seed = (uint32_t *)g_seed;

hal_aes_deinit(&g_aes_handle);
hal_aes_init(&g_aes_handle);
```

   A code snippet of the AES module in CBC mode is as follows:

```
g_aes_handle.p_instance         = AES;
g_aes_handle.init.key_size      = AES_KEYSIZE_128BITS;
g_aes_handle.init.p_key         = (uint32_t *)g_key128_cbc;
g_aes_handle.init.chaining_mode = AES_CHAININGMODE_CBC;
g_aes_handle.init.p_init_vector = (uint32_t *)g_iv_cbc;
g_aes_handle.init.dpa_mode = DISABLE;
g_aes_handle.init.p_seed = (uint32_t *)g_seed;

hal_aes_deinit(&g_aes_handle);
hal_aes_init(&g_aes_handle);
```

- init.key_size: This parameter sets the key size. Options: AES_KEYSIZE_128BITS, AES_KEYSIZE_192BITS, and AES_KEYSIZE_256BITS. In this example, init.key_size is set to AES_KEYSIZE_128BITS, indicating a 128-bit key is used.
- init.p_key: encryption/decryption key, provided by users
- init.chaining_mode: This parameter sets the encryption/decryption mode. Options: AES_CHAININGMODE_ECB and AES_CHAININGMODE_CBC
- init.p_init_vector: This parameter initializes vectors in CBC mode. No configuration on this parameter is required in ECB mode.
- dpa_mode: This parameter enables or disables the differential power analysis (DPA) function.
- p_seed: a random seed provided by users

2. Call AES encryption and decryption APIs.

   In this part, ECB mode is used as an example. For encryption and decryption in CBC mode, refer to the ECB mode.

   - Perform encryption/decryption in polling mode.

```
hal_aes_ecb_encrypt(&g_aes_handle, (uint32_t *)g_plaintext_ecb, sizeof(g_plaintext_ecb),
 (uint32_t *)g_encrypt_result, 5000)
hal_aes_ecb_decrypt(&g_aes_handle, (uint32_t *)g_encrypt_result, sizeof(g_encrypt_result),
 (uint32_t *)g_decrypt_result, 5000)
```

   (1). Call hal_aes_ecb_encrypt() to encrypt plaintexts in polling mode. The API execution results are returned in the g_encrypt_result parameter when the encryption completes.

   (2). Call hal_aes_ecb_decrypt() API to decrypt ciphertexts in polling mode. The API execution results are returned in the g_decrypt_result parameter when the decryption completes.

   - Perform encryption/decryption in non-polling mode (interrupt mode).

```
hal_aes_ecb_encrypt_it(&g_aes_handle, (uint32_t *)g_plaintext_ecb, sizeof(g_plaintext_ecb),
 (uint32_t *)g_encrypt_result)
hal_aes_ecb_decrypt_it(&g_aes_handle, (uint32_t *)g_encrypt_result,
 sizeof(g_encrypt_result), (uint32_t *)g_decrypt_result)

void hal_aes_done_callback(aes_handle_t *haes)
{
    g_int_done_flag = 1;
}

void hal_aes_error_callback(aes_handle_t *haes)
{
    printf("\r\nGet an Error!\r\n");
}
```

   (1). Call hal_aes_ecb_encrypt_it() to encrypt plaintexts in interrupt mode. The hal_aes_done_callback() or hal_aes_error_callback() returns the encryption results. Users can customize the APIs on demand.

   (2). Call hal_aes_ecb_decrypt_it() to decrypt plaintexts in interrupt mode. The hal_aes_done_callback() or hal_aes_error_callback() returns the decryption results. Users can customize the APIs on demand.

3.2.1.2 Test and Verification

1. Download *aes_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the AES encryption/decryption results in the **Receive Data** pane on GRUart.

3.3 AON_GPIO

The AON_GPIO is a General-Purpose Input/Output port module that works based on low-speed clocks. The module can be used as input, output, or for other special functions (such as waking up an MCU in sleep mode or triggering interrupts).

This section focuses on how to use the AON_GPIO module as input & output as well as wakeup source.

3.3.1 AON_GPIO Input & Output

The AON_GPIO Input & Output example implements input and output of the AON_GPIO module.

- When the AON_GPIO module is set as general-purpose output, the module can control the I/O level changes.
- When the AON_GPIO module is set as general-purpose input, the module can read the I/O levels of the pins.

The source code and project file of the AON_GPIO Input & Output example are in `SDK_Folder\projects\peripheral\aon_gpio\aon_gpio_output_input`, and the project file is in the Keil_5 folder.

3.3.1.1 Code Interpretation

Figure 3-7 shows the workflow of an AON_GPIO Input & Output example project.

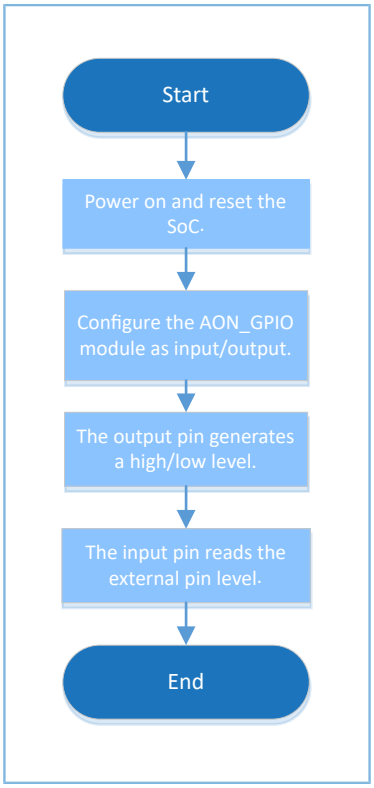

Figure 3-7 AON_GPIO Input & Output example workflow

1. Configure the AON_GPIO mode as input/output.

```
#define AON_GPIO_DEFAULT_CONFIG
{
     .pin      = AON_GPIO_PIN_ALL,
     .mode     = AON_GPIO_MODE_INPUT,
     .pull     = AON_GPIO_PULLDOWN,
     .mux      = AON_GPIO_MUX_7,
}

aon_gpio_init_t aon_gpio_init = AON_GPIO_DEFAULT_CONFIG;

aon_gpio_init.pin  = AON_GPIO_PIN_7;
aon_gpio_init.mode = AON_GPIO_MODE_OUTPUT;
hal_aon_gpio_init(&aon_gpio_init);

aon_gpio_init.pin  = AON_GPIO_PIN_6;
aon_gpio_init.mode = AON_GPIO_MODE_INPUT;
hal_aon_gpio_init(&aon_gpio_init);
```

- init.pin: This parameter sets an ID for a pin. Options: any combination of AON_GPIO_PIN_0 to AON_GPIO_PIN_7
- init.mode: This parameter sets the pin operating mode. Options: AON_GPIO_MODE_INPUT, AON_GPIO_MODE_OUTPUT, AON_GPIO_MODE_MUX, AON_GPIO_MODE_IT_RISING, AON_GPIO_MODE_IT_FALLING, AON_GPIO_MODE_IT_HIGH, and AON_GPIO_MODE_IT_LOW

- init.pull: This parameter configures a pull-up/pull-down resistor. Options: AON_GPIO_NOPULL, AON_GPIO_PULLUP, and AON_GPIO_PULLDOWN
- init.mux: This parameter configures Pin Mux. For details, see Pin Mux tables in "Pin Mux" in *GR551x Datasheet* . Set this parameter to AON_GPIO_MUX_7 when configuring the AON_GPIO mode as input/output.

When setting the module as input, modify the mode as described below:

```
aon_gpio_init.mode = AON_GPIO_MODE_INPUT;
```

2. Set the output level of a pin.

```
hal_aon_gpio_write_pin(AON_GPIO_PIN_7, AON_GPIO_PIN_RESET);
```

The parameter AON_GPIO_PIN_RESET means the pin level is set to low, and AON_GPIO_PIN_SET means the pin level is set to high.

3. Read the input level of a pin.

```
pin_level = hal_aon_gpio_read_pin(AON_GPIO_PIN_6);
```

Setting pin_level to 0 means the pin is at a low level while setting pin_level to 1 means the pin is at a high level.

3.3.1.2 Test and Verification

1. Download *aon_gpio_io_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the input/output status of the pin in the **Receive Data** pane on GRUart.

3.3.2 AON_GPIO Wakeup

The AON_GPIO Wakeup example detects interrupts of the AON_GPIO module.

When the AON_GPIO module is set to external input interrupt mode with interrupt enabled, an interrupt handler is triggered by changes on I/O pin levels. The AON_GPIO module can be configured as a wakeup source to wake up SoCs in sleep mode.

The source code and project file of the AON_GPIO Wakeup example are in SDK_Folder\projects\peripheral\aon_gpio\aon_gpio_wakeup, and the project file is in the Keil_5 folder.

3.3.2.1 Code Interpretation

Figure 3-8 and Figure 3-9 show the workflows of an AON_GPIO Wakeup example project.

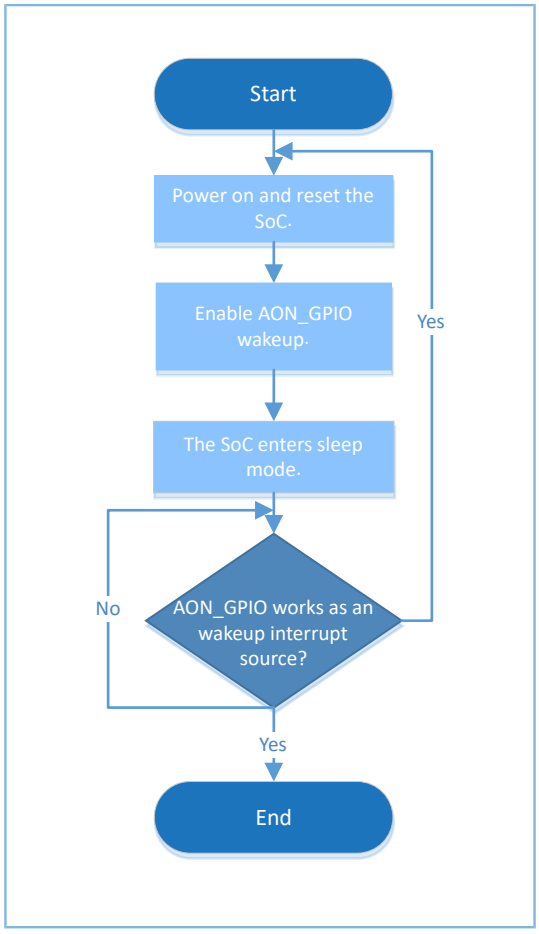

Figure 3-8 AON_GPIO Wakeup example workflow (sleep mode)

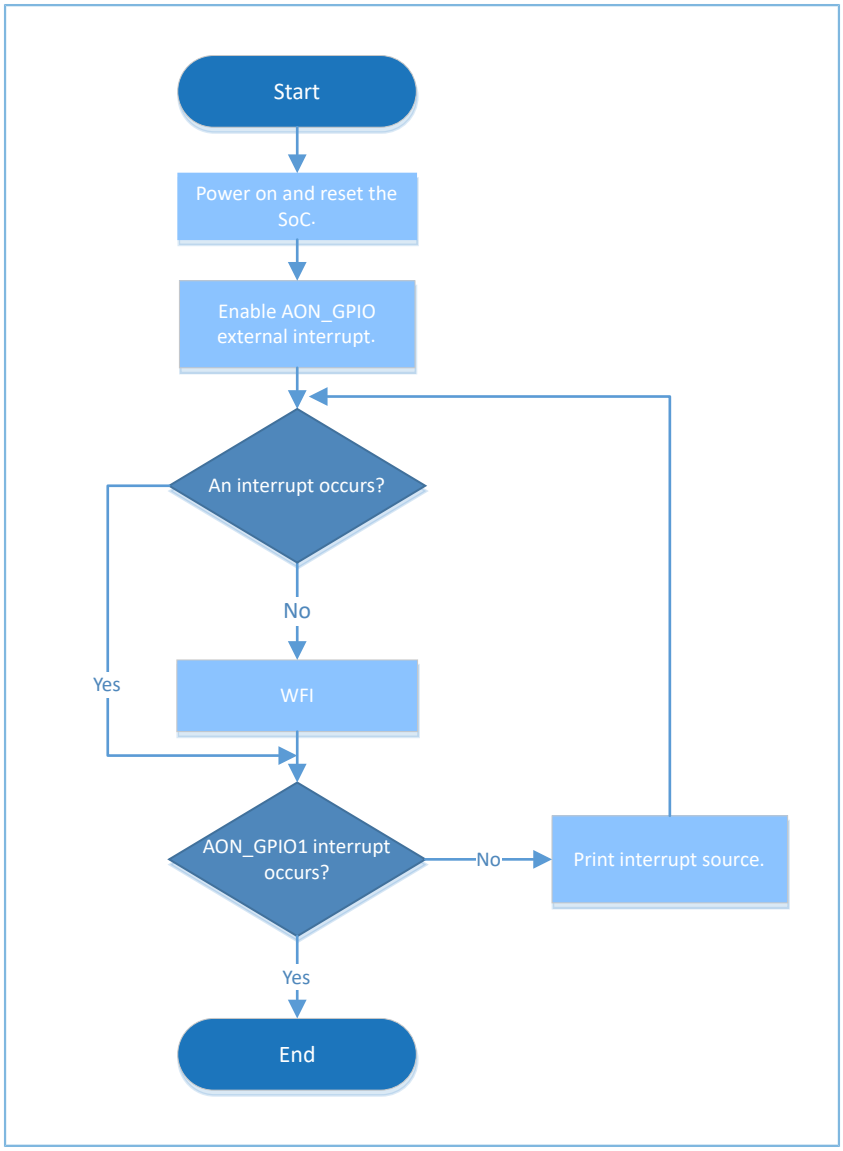

Figure 3-9 AON_GPIO Wakeup example workflow (WFI/WFE)

1. Configure an external interrupt for the AON_GPIO module.

```
aon_gpio_init_t aon_gpio_init = AON_GPIO_DEFAULT_CONFIG;
aon_gpio_init.pin  = KEY_OK_PIN;
aon_gpio_init.mode = KEY_ANO_TRIGGER_MODE;
aon_gpio_config.pull = AON_GPIO_PULLUP;
hal_aon_gpio_init(&aon_gpio_init);
```

   For more information about AON_GPIO configurations, see “Section 3.3.1 AON_GPIO Input & Output”. In this example, set aon_gpio_init.pin to KEY_OK_PIN, which means AON_GPIO_PIN_1 is selected.

2. Clear and enable AON_GPIO interrupt. The code snippet is as follows:

```
hal_nvic_clear_pending_irq(EXT2_IRQn);
hal_nvic_enable_irq(EXT2_IRQn);
```

3. The SoC enters sleep mode. The code snippet is as follows:

```
while (!g_exit_flag)
{
    printf("\r\nEnter sleep.\r\n");
    SCB->SCR |= 0x04;
    __WFI();
    printf("Wakeup from sleep.\r\n");
}
```

4. Press **OK** on the GR5515 SK Board to wake up the SoC and end the wakeup routine.

3.3.2.2 Test and Verification

1. Download *aon_gpio_wakeup_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the information of an SoC awoken by AON_GPIO or sleep mode debugging information of the board in the **Receive Data** pane on GRUart.

3.4 GPIO

The General-Purpose Input/Output (GPIO) port module works based on system clocks. The module can be used as input, output, or for other special functions (such as waking up an MCU in sleep mode or triggering interrupts). In general, the module is used to control peripherals or detect the input status of peripherals in interrupt mode.

This section focuses on how to use the GPIO module as Input & Output, Interrupt, LED, as well as Wakeup.

3.4.1 GPIO Input & Output

The GPIO Input & Output example implements input and output of the GPIO module.

- When the GPIO module is set as general-purpose output, the module can control the I/O level changes.
- When the GPIO module is set as general-purpose input, the module can read the I/O levels of the pins.

The source code and project file of the GPIO Input & Output example are in SDK_`Folder\projects\peripheral\gpio\gpio_output_input`, and the project file is in the Keil_5 folder.

3.4.1.1 Code Interpretation

Figure 3-10 shows the workflow of a GPIO Input & Output example project.

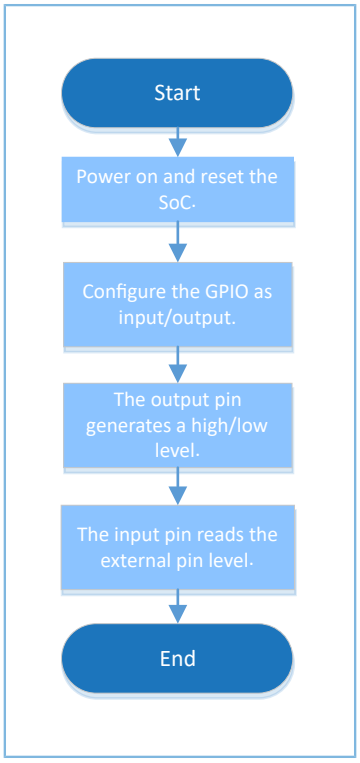

Figure 3-10 GPIO Input & Output example workflow

1. Configure the GPIO mode as input/output.

```
#define GPIO_DEFAULT_CONFIG                         \
{                                                   \
      .pin         = GPIO_PIN_ALL,                  \
      .mode        = GPIO_MODE_INPUT,               \
      .pull        = GPIO_PULLDOWN,                 \
      .mux         = GPIO_PIN_MUX_GPIO,             \
}

gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;
gpio_config.mode = GPIO_MODE_OUTPUT;
gpio_config.pin  = GPIO_PIN_12;
hal_gpio_init(GPIO1, &gpio_config);
```

- pin: This parameter sets an ID for a pin. Options: any combination of values from GPIO_PIN_0 to GPIO_PIN_15
- mode: This parameter sets the pin operating mode. Options: GPIO_MODE_INPUT, GPIO_MODE_OUTPUT, GPIO_MODE_MUX, GPIO_MODE_IT_RISING, GPIO_MODE_IT_FALLING, GPIO_MODE_IT_HIGH, and GPIO_MODE_IT_LOW
- pull: This parameter configures a pull-up/pull-down resistor. Options: GPIO_NOPULL, GPIO_PULLUP, and GPIO_PULLDOWN.
- mux: This parameter configures Pin Mux. For details, see Pin Mux tables in “Pin Mux” in *GR551x Datasheet*. Set this parameter to GPIO_MUX_7 when configuring the GPIO mode as input/output.

When setting the module as input, modify the mode as described below:

```
gpio_init.mode = GPIO_MODE_INPUT;
```

2. Set the output level of a pin.

```
hal_gpio_write_pin(GPIO1, GPIO_PIN_12, GPIO_PIN_RESET);
```

   The parameter GPIO_PIN_RESET means the pin level is set to low, and GPIO_PIN_SET means the pin level is set to high.

3. Read the input level of a pin.

```
pin_level = hal_gpio_read_pin(GPIO1, GPIO_PIN_13);
```

   Setting pin_level to 0 means the pin is at a low level while setting pin_level to 1 means the pin is at a high level.

3.4.1.2 Test and Verification

1. Download *gpio_io_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the input/output status of the pin in the **Receive Data** pane on GRUart.

3.4.2 GPIO Interrupt

The GPIO Interrupt example enables a GPIO pin to work as interrupt input. Such interrupts can be triggered by both edge transition (rising/falling edge) and level change (high/low level).

The source code and project file of the GPIO Interrupt example are in SDK_Folder\projects\peripheral\gpio\gpio_interrupt, and the project file is in the Keil_5 folder.

3.4.2.1 Code Interpretation

Figure 3-11 shows the workflow of a GPIO Interrupt example project.

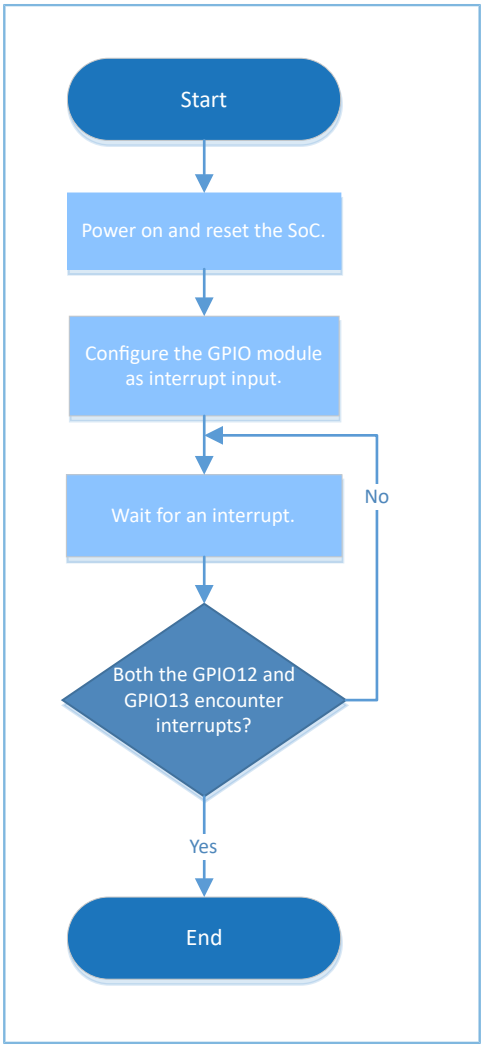

Figure 3-11 GPIO Interrupt example workflow

1. Configure the GPIO module as interrupt input.

```
gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;
gpio_config.mode = GPIO_MODE_IT_FALLING;
gpio_config.pin  = GPIO_KEY0 | GPIO_KEY1;
gpio_config.pull = GPIO_PULLUP;
hal_gpio_init(GPIO_KEY_PORT, &gpio_config);

/* Enable interrupt */
hal_nvic_clear_pending_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
hal_nvic_enable_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
```

   For more information about GPIO configurations, see "Section 3.4.1 GPIO Input & Output". Set mode to GPIO_MODE_IT_FALLING and pin to GPIO_KEY0 | GPIO_KEY1 (corresponding to GPIO_PIN_12 or GPIO_PIN_13)

2. Call hal_nvic_clear_pending_irq() and hal_nvic_enable_irq() to clear and enable GPIO interrupt. The code snippet is as follows:

```
hal_nvic_clear_pending_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
hal_nvic_enable_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
```

3. The hal_gpio_exti_callback() callback function returns the interrupt status. Users can redefine this function on demand.

3.4.2.2 Test and Verification

1. Download *gpio_interrupt_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the GPIO interrupt input results in the **Receive Data** pane on GRUart.

3.4.3 GPIO LED

The GPIO LED example enables a GPIO pin as output to drive an LED.

The source code and project file of the GPIO LED example are in SDK_Folder\projects\peripheral\gpio\gpio_led, and the project file is in the Keil_5 folder.

3.4.3.1 Code Interpretation

1. Configure the GPIO mode as output.

```
gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;
gpio_config.mode = GPIO_MODE_OUTPUT;
gpio_config.pin  = LED2_PIN;
hal_gpio_init(LED2_PORT, &gpio_config);
```

   - For more information about GPIO configurations, see "Section 3.4.1 GPIO Input & Output". Set pin to LED2_PIN, corresponding to GPIO_PIN_4.
   - The parameter mux configures Pin Mux. For details, see Pin Mux tables in "Pin Mux" in *GR551x Datasheet*. Set this parameter to GPIO_MUX_7 when configuring the GPIO mode as input/output.

2. Call the LED2_TOG() macro to implement hal_gpio_toggle_pin() to toggle the level of a GPIO pin. The code snippet is as follows:

```
#define LED2_TOG()          hal_gpio_toggle_pin(LED2_PORT, LED2_PIN)
```

3.4.3.2 Test and Verification

1. Download *gpio_led_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the GPIO-driven LED result and the LED status (on/off) in the **Receive Data** pane on GRUart.

3.4.4 GPIO Wakeup

The GPIO Wakeup example enables a GPIO pin to work as external interrupt input that wakes an SoC up from WFI or WFE.

When the GPIO module is set to external input interrupt mode with interrupt enabled, an interrupt handler is triggered by edge transition (rising/falling edge) and level change (high/low level).

The source code and project file of the GPIO Wakeup example are in SDK_Folder\projects\peripheral\gpio\gpio_wakeup, and the project file is in the Keil_5 folder.

3.4.4.1 Code Interpretation

Figure 3-12 shows the workflow of a GPIO Wakeup example project.

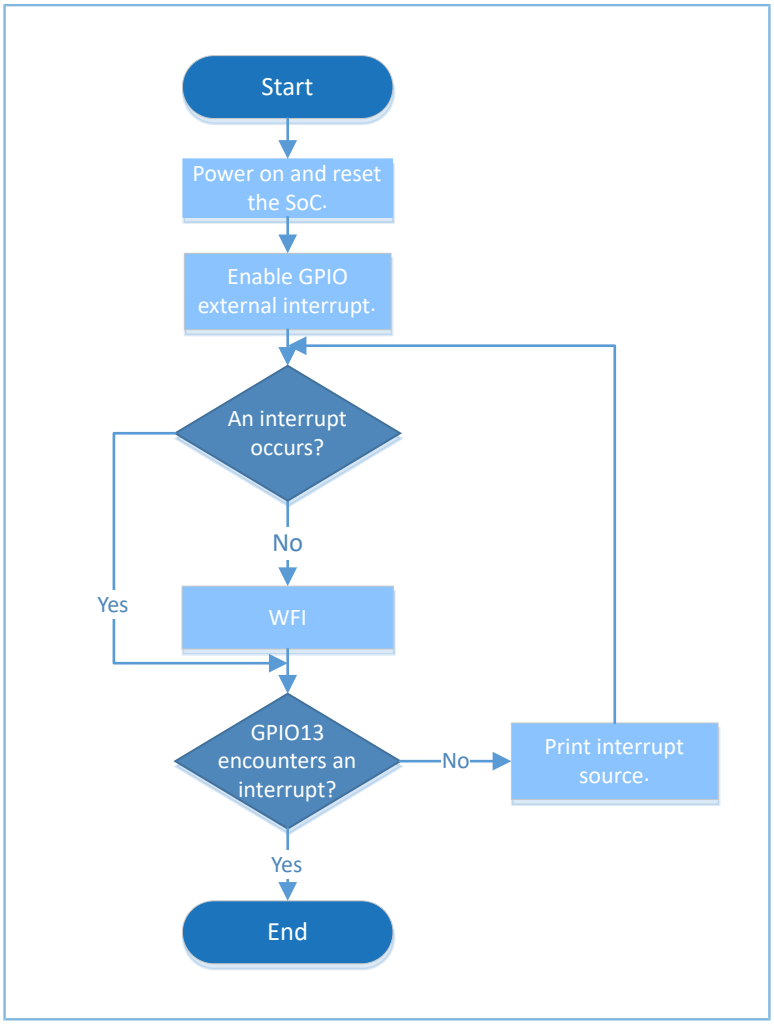

Figure 3-12 GPIO Wakeup example workflow (from WFI/WFE)

1. Configure the GPIO mode as external interrupt.

```
gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;
gpio_config.mode = GPIO_MODE_IT_FALLING;
gpio_config.pin  = GPIO_KEY0 | GPIO_KEY1;
gpio_config.pull = GPIO_PULLUP;
hal_gpio_init(GPIO_KEY_PORT, &gpio_config);
```

- For more information about GPIO configurations, see "Section 3.4.1 GPIO Input & Output". Set mode to GPIO_MODE_IT_FALLING and pin to GPIO_KEY0 | GPIO_KEY1 (corresponding to GPIO_PIN_12 or GPIO_PIN_13).
- The parameter mux configures Pin Mux. For details, see Pin Mux tables in "Pin Mux" in *GR551x Datasheet*. Set this parameter to GPIO_MUX_7 when configuring the GPIO mode as external interrupt input.

2. Call hal_nvic_clear_pending_irq() and hal_nvic_enable_irq() to clear and enable GPIO interrupt. The code snippet is as follows:

```
hal_nvic_clear_pending_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
hal_nvic_enable_irq(GPIO_GET_IRQNUM(GPIO_KEY_PORT));
```

3. The SoC enters sleep mode. The code snippet is as follows:

```
while (!g_exit_flag)
{
    printf("\r\nEnter sleep at counter = %d\r\n", sleep_count);
    SCB->SCR |= 0x04;
    __WFI();
    printf("Wakeup from sleep at counter = %d\r\n", sleep_count++);
}
```

4. Press **UP** and **DOWN** on the board to select GPIO_KEY0 to wake up the SoC and GPIO_KEY1 to wake up the SoC and exit the example.

3.4.4.2 Test and Verification

1. Download *gpio_wakeup_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the information of an SoC CPU awoken by a GPIO pin from WFI/WFE in the **Receive Data** pane on GRUart.

3.5 HMAC

The Hash-based Message Authentication Code (HMAC) is a specific type of message authentication code involving a one-way hash function and a secret cryptographic key, making it applicable to authenticate message digests and signatures.

3.5.1 HMAC

The HMAC example supports two digest generation modes: SHA-256 and HMAC-SHA256.

- The SHA-256 algorithm generates a 256-bit hash value (known as message digest) based on input messages. Users need to input the message in compliance with HMAC API-related regulations.
- The HMAC-SHA256 algorithm generates a message digest based on input keys and messages by using a hash function. The input key shall be 256 bits in size.

The source code and project file of the HMAC example are in SDK_Folder\projects\peripheral\hmac\hmac, and the project file is in the Keil_5 folder.

This section focuses on how the HMAC example authenticates messages in interrupt, query, and DMA modes.

3.5.1.1 Code Interpretation

Figure 3-13 and Figure 3-14 show the workflows of an HMAC example project using SHA-256 and HMAC-SHA256 respectively.

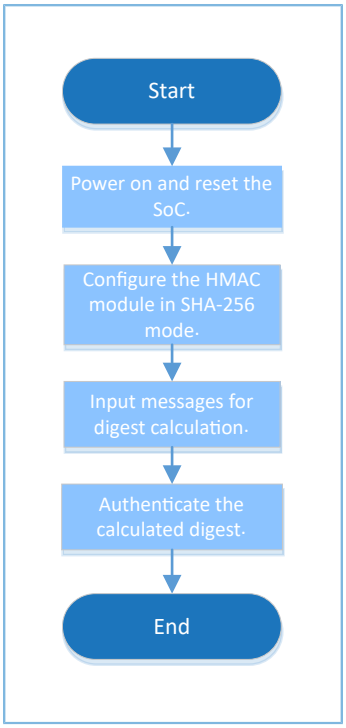

Figure 3-13 HMAC example workflow (SHA-256)

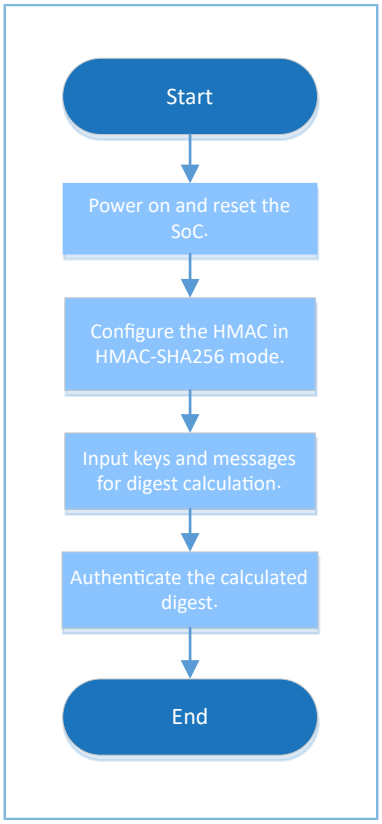

Figure 3-14 HMAC example workflow (HMAC-SHA256)

1. Configure the HMAC module.

```
g_hmac_handle.p_instance       = HMAC;
g_hmac_handle.init.mode        = HMAC_MODE_HMAC;
g_hmac_handle.init.p_key       = (uint32_t *)hmac_key;
g_hmac_handle.init.p_user_hash = NULL;
g_hmac_handle.init.dpa_mode    = DISABLE;
g_hmac_handle.init.key_fetch_type = HAL_HMAC_KEYTYPE_MCU;
g_hmac_handle.init.enable_irq = HAL_HMAC_DISABLE_IRQ;
g_hmac_handle.init.enable_dma_mode = HAL_HMAC_DISABLE_DMA;
```

- init.mode: This parameter sets an HMAC operation mode. Options: HMAC_MODE_SHA (when applying SHA-256) and HMAC_MODE_HMAC (when applying HMAC-SHA256).
- init.p_key: This parameter sets the key and is valid only in HMAC-SHA256 mode.
- init.p_user_hash: This parameter represents a user-defined initial hash value. Users can define this value on demand.
- init.dpa_mode: This parameter enables or disables the DPA function.
- init.key_fetch_type: the key source. Options: HAL_HMAC_KEYTYPE_MCU, HAL_HMAC_KEYTYPE_AHB, and HAL_HMAC_KEYTYPE_KRAM.
- init.enable_irq: This parameter enables or disables an interrupt request (IRQ). Options: HAL_HMAC_ENABLE_IRQ and HAL_HMAC_DISABLE_IRQ.

- init.enable_dma_mode: This parameter enables or disables the DMA mode. Options: HAL_HMAC_ENABLE_DMA and HAL_HMAC_DISABLE_DMA.

2. Users can configure the polling mode, interrupt mode, or DMA mode on demand. Details on the configuration are provided as follows:

   (1). Reload the customized initial hash value p_user_hash and the key p_key (only effective when applying HMAC-SHA256) on demand.

   (2). Configure the modes on demand.

   - init.enable_irq: HAL_HMAC_DISABLE_IRQ (for polling and DMA modes) or HAL_HMAC_ENABLE_IRQ (for interrupt mode)
   - init.enable_dma_mode: HAL_HMAC_DISABLE_DMA (for polling and interrupt modes) or HAL_HMAC_ENABLE_DMA (for DMA mode)
   - init.mode: HMAC_MODE_SHA or HMAC_MODE_HMAC

   (3). Compute message digests with hal_hmac_sha256_digest(). If a data stream is too long to be calculated for once, divide the stream into multiple segments, and reload the customized initial hash value p_user_hash from the previous calculation in all but the first segment.

   While in interrupt or DMA mode, hal_hmac_done_callback() will be called when the calculation completes, and hal_hmac_error_callback() will be called when an error occurs. In the latter case, users shall rewrite hal_hmac_error_callback().

3.5.1.2 Test and Verification

1. Download *hmac_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the HMAC calculation results in the **Receive Data** pane on GRUart.

3.6 I2C

The Inter-integrated Circuit (I2C) is a kind of simple, bidirectional 2-wire synchronous serial bus. It requires only two wires to transfer information between devices connected to the bus. In general, it is used to get data of sensors, for example: temperature sensor and acceleration sensor.

The sections below focus on how to read/write data in interrupt, query, and DMA modes through the I2C interface and how to use the master/slave APIs.

3.6.1 I2C DMA UART

The I2C DMA UART example in a GR551x SDK shows how I2C transmits data received from UART in DMA mode.

The source code and project file of the I2C DMA UART example are in `SDK_Folder\projects\peripheral\i2c\i2c_dma_uart`, and project file is in the Keil_5 folder.

3.6.1.1 Code Interpretation

Figure 3-15 shows the workflow of the example project.

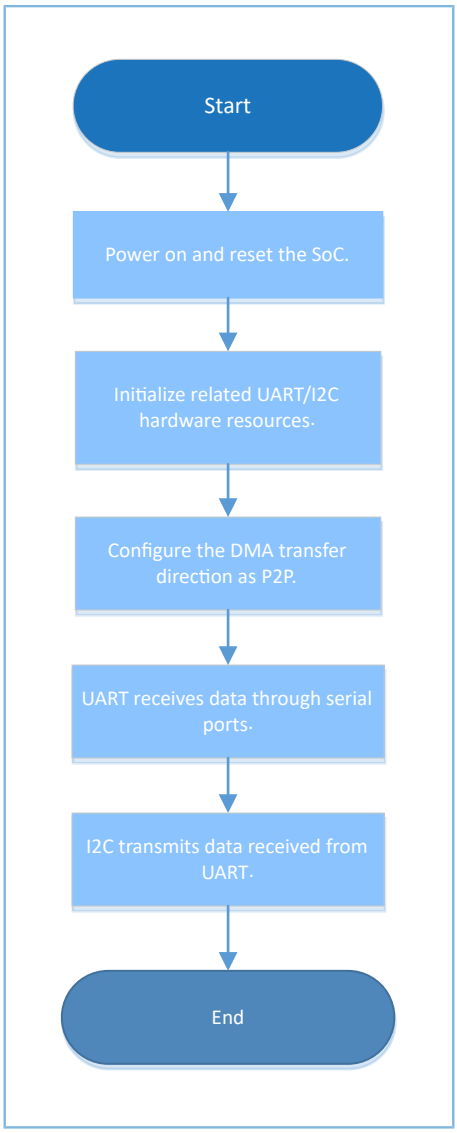

Figure 3-15 I2C DMA UART example workflow

1. Configure the I2C module.

```
p_i2c_handle.p_instance             = I2C_MODULE;
p_i2c_handle.init.speed             = I2C_SPEED_400K;
p_i2c_handle.init.own_address       = 0x55;
p_i2c_handle.init.addressing_mode   = I2C_ADDRESSINGMODE_7BIT;
p_i2c_handle.init.general_call_mode = I2C_GENERALCALL_DISABLE;

hal_i2c_deinit(&p_i2c_handle);
hal_i2c_init(&p_i2c_handle);
```

- init.speed: I2C transfer speed. Options: I2C_SPEED_100K, I2C_SPEED_400K, I2C_SPEED_1000K, and I2C_SPEED_2000K

- init.own_address: 7-bit or 10-bit I2C local address that can be customized by users
- init.addressing_mode: I2C addressing mode. Options: I2C_ADDRESSINGMODE_7BIT and I2C_ADDRESSINGMODE_10BIT
- init.general_call_mode: I2C general call addressing mode. Options: I2C_GENERALCALL_DISABLE and I2C_GENERALCALL_ENABLE

2. Initialize the I2C hardware.

```
void hal_i2c_msp_init(i2c_handle_t * p_i2c)
{
    gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;

    gpio_config.mode = GPIO_MODE_MUX;
    gpio_config.pull = GPIO_PULLUP;
    gpio_config.pin  = I2C_SCL_PIN | I2C_SDA_PIN;
    gpio_config.mux  = I2C_GPIO_MUX;
    hal_gpio_init(I2C_GPIO_PORT, &gpio_config); (1)

    __HAL_LINKDMA(p_i2c, p_dmatx, s_dma_handle);
    /* Configure the DMA handler for Transmission process */
    hi2c->p_dmatx->channel                 = DMA_Channel0;
    hi2c->p_dmatx->init.direction           = DMA_PERIPH_TO_PERIPH; (2)
    hi2c->p_dmatx->init.src_request         = DMA_REQUEST_UART0_RX; (3)
    hi2c->p_dmatx->init.dst_request          = DMA_REQUEST_I2C0_TX; (4)
    hi2c->p_dmatx->init.src_increment        = DMA_SRC_NO_CHANGE; (5)
    hi2c->p_dmatx->init.dst_increment        = DMA_DST_NO_CHANGE; (6)
    hi2c->p_dmatx->init.src_data_alignment  = DMA_SDATAALIGN_BYTE; (7)
    hi2c->p_dmatx->init.dst_data_alignment  = DMA_DDATAALIGN_BYTE; (8)
    hi2c->p_dmatx->init.mode                   = DMA_NORMAL;
    hi2c->p_dmatx->init.priority              = DMA_PRIORITY_LOW;

    hal_dma_deinit(p_i2c->p_dmatx);
    hal_dma_init(p_i2c->p_dmatx);
}
```

Notes:

(1): Initialize I/O pins of the hardware, and configure the pins mapped with the I2C module into I2C mode.

(2): Configure the DMA transfer direction as P2P.

(3)–(4): Configure DMA channel source and destination as UART RX and I2C TX respectively.

(5)–(6): The pattern of both DMA channel source address and destination address is fixed.

(7)–(8): The transmission unit for DMA channels is 8-bit wide.

3. Configure the slave address.

Before transmission, define the I2C of the GR5515 SK Board as the master, and configure the slave address. The code snippet is as follows:

```
/* Enable Master Mode and Set Slave Address */
ll_i2c_disable(p_i2c_handle->p_instance);
ll_i2c_enable_master_mode(p_i2c_handle->p_instance);
ll_i2c_set_slave_address(p_i2c_handle->p_instance, SLAVE_DEV_ADDR);
ll_i2c_enable(p_i2c_handle->p_instance);
```

4. Set the UART reception threshold and the DMA burst length. The code snippet is as follows:

```
ll_i2c_set_dma_tx_data_level(p_i2c_handle->p_instance, 4U);
ll_uart_set_rx_fifo_threshold(LOG_UART_GRP,   LL_UART_RX_FIFO_TH_CHAR_1);
ll_dma_set_source_burst_length(DMA, p_i2c_handle->p_dmatx->channel,
 LL_DMA_SRC_BURST_LENGTH_1);
ll_dma_set_destination_burst_length(DMA, p_i2c_handle->p_dmatx->channel,
 LL_DMA_DST_BURST_LENGTH_4);
```

5. Wait until UART receives data. The code snippet is as follows:

```
/* Wait until receive any data */
while(!ll_uart_is_active_flag_rfne(SERIAL_PORT_GRP));
```

6. Start DMA transmission, and wait until the transmission is completed. The code snippet is as follows:

```
hal_dma_start(p_i2c_handle->p_dmatx, (uint32_t)&UART0->RBR_DLL_THR, (uint32_t)&
 p_i2c_handle->p_instance->DATA_CMD, TEST_LENGTH);
/* Enable DMA Request */
ll_i2c_enable_dma_req_tx(p_i2c_handle->p_instance);
hal_dma_poll_for_transfer(p_i2c_handle->p_dmatx, 1000);
```

7. The STOP signal shall be enabled for transferring the last byte of I2C data. Therefore, the STOP process shall be added after the DMA transmission is completed. The code snippet is as follows:

```
/* Disable DMA Request */
ll_i2c_disable_dma_req_tx(p_i2c_handle->p_instance);
while(RESET == ll_i2c_is_active_flag_status_tfnf(p_i2c_handle->p_instance));
ll_i2c_transmit_data8(p_i2c_handle->p_instance, 0, LL_I2C_CMD_MST_WRITE |
 LL_I2C_CMD_MST_GEN_STOP);
```

3.6.1.2 Test and Verification

1. Download *i2c_dma_uart_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the implementation results after the I2C transfers data received from UART in DMA mode in the **Receive Data** pane on GRUart.

3.6.2 I2C ADXL345

The I2C ADXL345 example in a GR551x SDK implements communications between I2C drivers and I2C devices. Data is sent to the ADXL345 sensor and read from ADXL345 through I2C.

The source code and project file of the I2C ADXL345 example are in SDK_Folder\projects\peripheral\i2c\i2c_master_adxl345, and project file is in the Keil_5 folder.

3.6.2.1 Code Interpretation

Figure 3-16 shows the workflow of the example project.

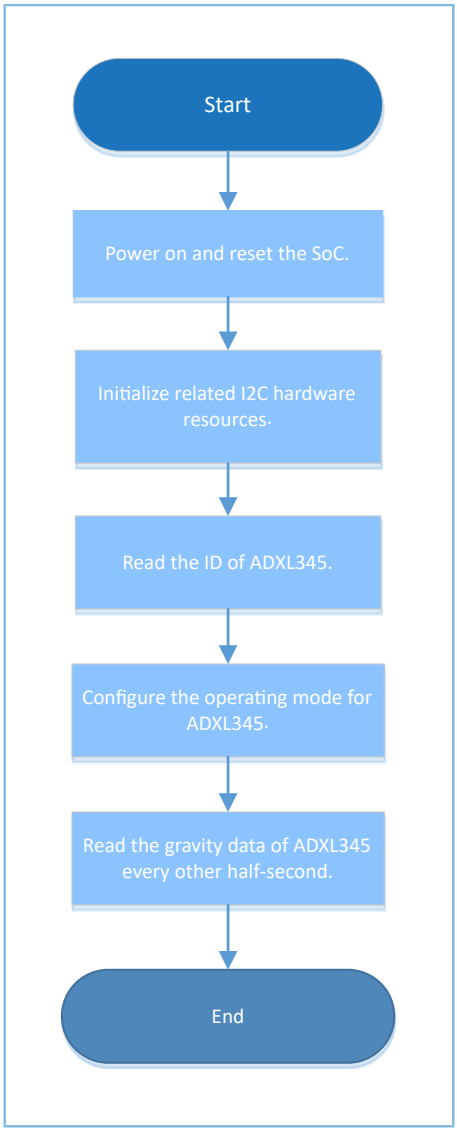

Figure 3-16 I2C ADXL345 example workflow

1. Configure the I2C module.

   For more information about I2C configurations, see "Section 3.6.1 I2C DMA UART". In this example project, speed is configured as I2C_SPEED_400K, addressing_mode as I2C_ADDRESSINGMODE_7BIT, and general_call_mode as I2C_GENERALCALL_DISABLE.

2. Initialize the I2C hardware.

```
void hal_i2c_msp_init(i2c_handle_t *p_i2c)
{
gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;

/*---------------------- Configure the I2C Pins ------------------------*/
    gpio_config.mode = GPIO_MODE_MUX;
    gpio_config.pull = GPIO_PULLUP;
    gpio_config.mux  = I2C_GPIO_MUX;
    gpio_config.pin  = I2C_SCL_PIN | I2C_SDA_PIN;
    hal_gpio_init(I2C_GPIO_PORT, &gpio_config);       (1)
```

```
    /*------------------- Configure the DMA for I2C -------------------*/
    /* Configure the DMA handler for Transmission process */
    s_dma_tx_handle.channel = DMA_Channel0;
    s_dma_tx_handle.init.src_request = DMA_REQUEST_MEM;
    s_dma_tx_handle.init.dst_request = p_i2c->p_instance == I2C0) ? DMA_REQUEST_I2C0_TX :
 DMA_REQUEST_I2C1_TX;
    s_dma_tx_handle.init.direction = DMA_MEMORY_TO_PERIPH;
    s_dma_tx_handle.init.src_increment = DMA_SRC_INCREMENT;    (2)
    s_dma_tx_handle.init.dst_increment = DMA_DST_NO_CHANGE;    (3)
    s_dma_tx_handle.init.src_data_alignment = DMA_SDATAALIGN_BYTE;   (4)
    s_dma_tx_handle.init.dst_data_alignment = DMA_DDATAALIGN_BYTE;   (5)
    s_dma_tx_handle.init.mode = DMA_NORMAL;
    s_dma_tx_handle.init.priority = DMA_PRIORITY_LOW;

    hal_dma_deinit(&s_dma_tx_handle);
    hal_dma_init(&s_dma_tx_handle);

/* Associate the initialized DMA handle to the I2C handle */
    __HAL_LINKDMA(p_i2c, p_dmatx, s_dma_tx_handle);      (6)

   /* Configure the DMA handler for reception process */
    s_dma_rx_handle.channel = DMA_Channel1;
    s_dma_rx_handle.init.src_request = (p_i2c->p_instance == I2C0) ? DMA_REQUEST_I2C0_RX :
 DMA_REQUEST_I2C1_RX;
    s_dma_rx_handle.init.dst_request = DMA_REQUEST_MEM;
    s_dma_rx_handle.init.direction = DMA_PERIPH_TO_MEMORY;
    s_dma_rx_handle.init.src_increment = DMA_SRC_NO_CHANGE;    (7)
    s_dma_rx_handle.init.dst_increment = DMA_DST_INCREMENT;    (8)
    s_dma_rx_handle.init.src_data_alignment = DMA_SDATAALIGN_BYTE;   (9)
    s_dma_rx_handle.init.dst_data_alignment = DMA_DDATAALIGN_BYTE;   (10)
    s_dma_rx_handle.init.mode = DMA_NORMAL;
    s_dma_rx_handle.init.priority = DMA_PRIORITY_LOW;

    hal_dma_deinit(&s_dma_rx_handle);
    hal_dma_init(&s_dma_rx_handle);

   /* Associate the initialized DMA handle to the the I2C handle */
    __HAL_LINKDMA(p_i2c, p_dmarx, s_dma_rx_handle);                    (11)

    /* NVIC for DMA */
    hal_nvic_set_priority(DMA_IRQn, 0, 1);
    hal_nvic_clear_pending_irq(DMA_IRQn);
    hal_nvic_enable_irq(DMA_IRQn);

    /* NVIC for I2C */
    hal_nvic_set_priority(I2C_GET_IRQNUM(p_i2c->p_instance), 0, 1);
    hal_nvic_clear_pending_irq(I2C_GET_IRQNUM(p_i2c->p_instance));
    hal_nvic_enable_irq(I2C_GET_IRQNUM(p_i2c->p_instance));}
```

Notes:

(1): Initialize I/O pins of the hardware, and configure the pins mapped with the I2C module into I2C mode.

(2)–(5): Configure the address increment mode and the transmission unit bit width for TX DMA channel source and destination. I2C TX channel source: arrays in RAM, with the address in increment mode; destination: I2C TX

FIFO register address, with the address not in increment mode. The transmission unit for I2C TX channels is 8-bit wide.

(6): Register TX DMA channel handles in I2C instance handles.

(7)–(10): Configure the address increment mode and the transmission unit width for RX DMA channel source and destination. I2C RX channel source: I2C RX FIFO register address, with the address not in increment mode; destination: arrays in RAM, with the address in increment mode. The transmission unit for I2C RX channels is 8-bit wide.

(11): Register RX DMA channel handles in I2C instance handles.

3. You need to implement read and write operations in I2C drivers based on requirements from different I2C devices. The code snippet is as follows:

```
void i2c_write_adxl345(uint8_t reg_addr, uint8_t *buf, uint8_t size)
{
    uint8_t wdata[256] = {0};

    wdata[0] = reg_addr;
    memcpy(&wdata[1], buf, size);
    hal_i2c_master_transmit(&g_i2c_handle, SLAVE_DEV_ADDR, wdata, size + 1, 5000);
}

void i2c_read_adxl345(uint8_t reg_addr, uint8_t *buf, uint8_t size)
{
    uint8_t wdata[1] = {0};

    wdata[0] = reg_addr;
    hal_i2c_master_transmit(&g_i2c_handle, SLAVE_DEV_ADDR, wdata, 1, 5000);
    hal_i2c_master_receive(&g_i2c_handle, SLAVE_DEV_ADDR, buf, size, 5000);
}
```

3.6.2.2 Test and Verification

1. Download *i2c_adxl345_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the implementation process about sending data to ADXL345, reading data from ADXL345, and printing ADXL345 data through I2C in the **Receive Data** pane on GRUart.

3.6.3 I2C Master & Slave

The I2C Master & Slave example in a GR551x SDK implements communications between I2C modules of the master and the slave.

The source code and project file of the I2C Master & Slave example are in `SDK_Folder\projects\peripheral\i2c\i2c_master_slave`, and project file is in the Keil_5 folder.

3.6.3.1 Code Interpretation

Figure 3-17 shows the workflow of the example project.

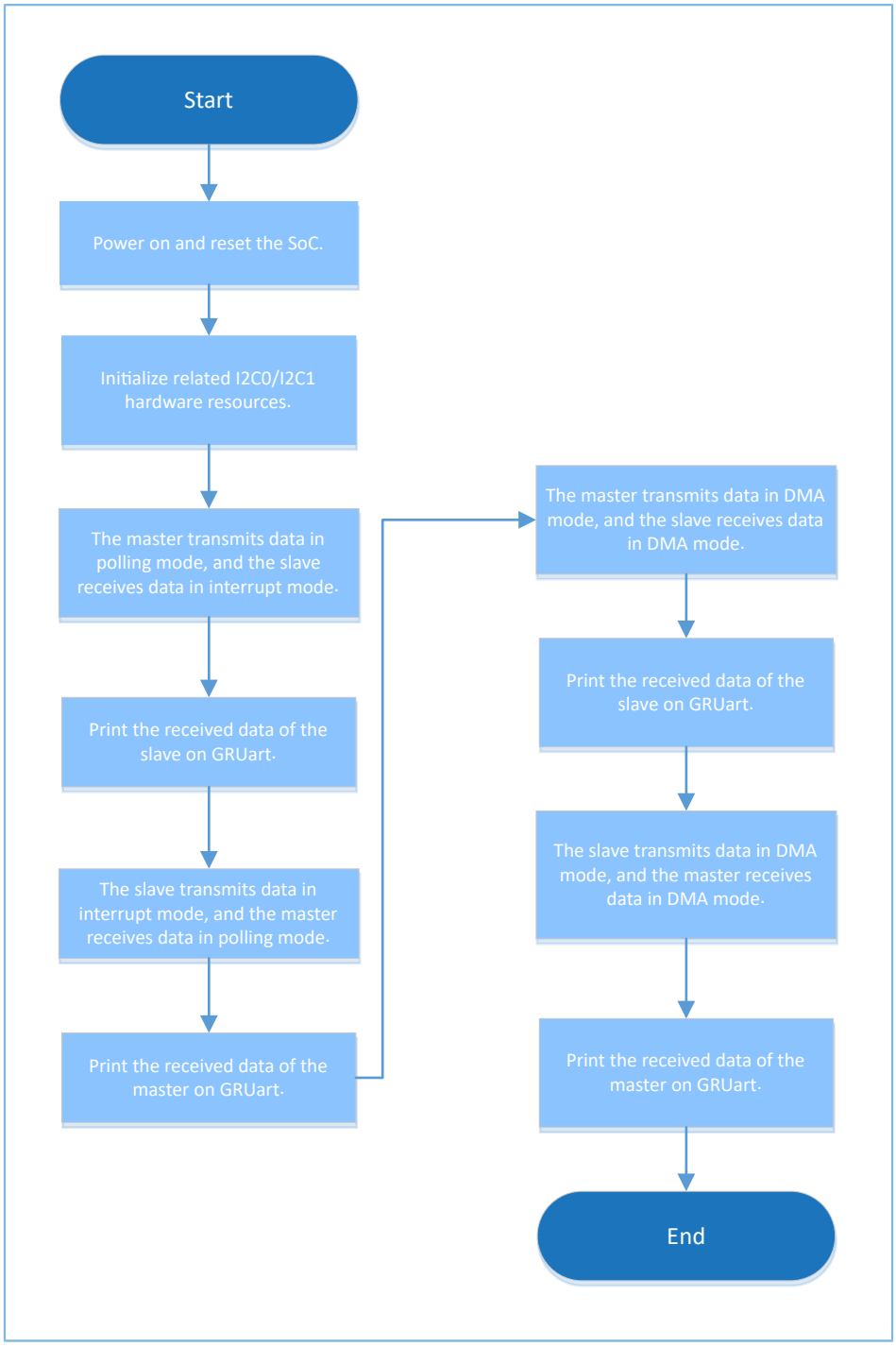

Figure 3-17 I2C Master & Slave example workflow

1. Configure the I2C module.

   For more information about I2C configurations, see "Section 3.6.1 I2C DMA UART". In this example project, speed is configured as I2C_SPEED_400K, addressing_mode as I2C_ADDRESSINGMODE_7BIT, and general_call_mode as I2C_GENERALCALL_DISABLE.

   Set the master/slave device addresses as follows:

```
#define MASTER_DEV_ADDR                 0x4D
#define SLAVE_DEV_ADDR                  0x55

g_i2cs_handle.init.own_address  = SLAVE_DEV_ADDR;

g_i2cm_handle.init.own_address = MASTER_DEV_ADDR;
```

2. Initialize the I2C hardware.

```
void hal_i2c_msp_init(i2c_handle_t *p_i2c)
{
    gpio_init_t gpio_config = GPIO_DEFAULT_CONFIG;
    msio_init_t msio_config = MSIO_DEFAULT_CONFIG;

    if (p_i2c->p_instance == I2C_MASTER_MODULE)
    {
        msio_config.pin  = I2C_MASTER_SCL_PIN | I2C_MASTER_SDA_PIN;
        msio_config.pull = MSIO_PULLUP;
        msio_config.mux  = I2C_MASTER_GPIO_MUX;
        hal_msio_init(&msio_config);
    }
    else if (p_i2c->p_instance == I2C_SLAVE_MODULE)
    {
       gpio_config.mode = GPIO_MODE_MUX;
       gpio_config.pull = GPIO_PULLUP;
       gpio_config.pin  = I2C_SLAVE_SCL_PIN | I2C_SLAVE_SDA_PIN;
       gpio_config.mux  = I2C_SLAVE_GPIO_MUX;
       hal_gpio_init(I2C_SLAVE_GPIO_PORT, &gpio_config);
    }
    NVIC_ClearPendingIRQ(I2C_GET_IRQNUM(p_i2c->p_instance));
    NVIC_EnableIRQ(I2C_GET_IRQNUM(p_i2c->p_instance));

    if (p_i2c->p_instance == I2C_MASTER_MODULE)
    {
        __HAL_LINKDMA(p_i2c, p_dmatx, s_i2cm_dma_tx_handle);
        __HAL_LINKDMA(p_i2c, p_dmarx, s_i2cm_dma_rx_handle);
        s_i2cm_dma_tx_handle.p_parent = p_i2c;
        s_i2cm_dma_rx_handle.p_parent = p_i2c;
        s_i2cm_dma_tx_handle.channel  = DMA_Channel0;
        s_i2cm_dma_rx_handle.channel  = DMA_Channel1;
    }
    else if (p_i2c->p_instance == I2C_SLAVE_MODULE)
    {
        __HAL_LINKDMA(p_i2c, p_dmatx, s_i2cs_dma_tx_handle);
        __HAL_LINKDMA(p_i2c, p_dmarx, s_i2cs_dma_rx_handle);
        s_i2cs_dma_tx_handle.p_parent = p_i2c;
        s_i2cs_dma_rx_handle.p_parent = p_i2c;
        s_i2cs_dma_tx_handle.channel  = DMA_Channel2;
        s_i2cs_dma_rx_handle.channel  = DMA_Channel3;
    }
    p_i2c->p_dmatx->init.src_request = DMA_REQUEST_MEM;
    p_i2c->p_dmarx->init.dst_request = DMA_REQUEST_MEM;
    if (p_i2c->p_instance == I2C0)
    {
        p_i2c->p_dmatx->init.dst_request = DMA_REQUEST_I2C0_TX;
        p_i2c->p_dmarx->init.src_request = DMA_REQUEST_I2C0_RX;
    }
    else if (p_i2c->p_instance == I2C1)
    {
        p_i2c->p_dmatx->init.dst_request = DMA_REQUEST_I2C1_TX;
```

```
        p_i2c->p_dmarx->init.src_request = DMA_REQUEST_I2C1_RX;
    }

    p_i2c->p_dmatx->init.direction           = DMA_MEMORY_TO_PERIPH;
    p_i2c->p_dmatx->init.src_increment       = DMA_SRC_INCREMENT;
    p_i2c->p_dmatx->init.dst_increment       = DMA_DST_NO_CHANGE;
    p_i2c->p_dmatx->init.src_data_alignment  = DMA_SDATAALIGN_BYTE;
    p_i2c->p_dmatx->init.dst_data_alignment  = DMA_DDATAALIGN_BYTE;
    p_i2c->p_dmatx->init.mode  = DMA_NORMAL;
    p_i2c->p_dmatx->init.priority  = DMA_PRIORITY_LOW;

    p_i2c->p_dmarx->init.direction           = DMA_PERIPH_TO_MEMORY;
    p_i2c->p_dmarx->init.src_increment       = DMA_SRC_NO_CHANGE;
    p_i2c->p_dmarx->init.dst_increment       = DMA_DST_INCREMENT;
    p_i2c->p_dmarx->init.src_data_alignment  = DMA_SDATAALIGN_BYTE;
    p_i2c->p_dmarx->init.dst_data_alignment  = DMA_DDATAALIGN_BYTE;
    p_i2c->p_dmarx->init.mode  = DMA_NORMAL;
    p_i2c->p_dmarx->init.priority  = DMA_PRIORITY_LOW;

    hal_dma_init(p_i2c->p_dmatx);
    hal_dma_init(p_i2c->p_dmarx);

    hal_nvic_clear_pending_irq(DMA_IRQn);
    hal_nvic_enable_irq(DMA_IRQn);
}
```

For more information about functions and parameters, see “Section 3.6.2.1 Code Interpretation”.

3. The slave receives data in interrupt mode, and the master sends data in polling mode. Due to the non-blocking mode adopted by the slave, the while loop is used to judge whether the slave has received the data in full in subsequent steps. The code snippet is as follows:

```
hal_i2c_slave_receive_it(&g_i2cs_handle, rdata, 256);
hal_i2c_master_transmit(&g_i2cm_handle, SLAVE_DEV_ADDR, wdata, 256, 5000);
while (hal_i2c_get_state(&g_i2cs_handle) ! = HAL_I2C_STATE_READY);
```

4. The slave sends data in interrupt mode, and the master receives data in polling mode. Due to the non-blocking mode adopted by the slave, the while loop is used to judge whether the slave has sent the data in full in subsequent steps. The code snippet is as follows:

```
hal_i2c_slave_transmit_it(&g_i2cs_handle, wdata, 256);
hal_i2c_master_receive(&g_i2cm_handle, SLAVE_DEV_ADDR, rdata, 256, 5000);
while (hal_i2c_get_state(&g_i2cs_handle) ! = HAL_I2C_STATE_READY);
```

5. Reception/Transmission operations for both the master and the slave are implemented in DMA mode. Due to the non-blocking mode adopted, the while loop is used to judge whether data reception or transmission is completed. The code snippet is as follows:

```
hal_i2c_slave_receive_dma(&g_i2cs_handle, rdata, 256);
hal_i2c_master_transmit_dma(&g_i2cm_handle, SLAVE_DEV_ADDR, wdata, 256);
while (hal_i2c_get_state(&g_i2cs_handle) ! = HAL_I2C_STATE_READY);

hal_i2c_slave_transmit_dma(&g_i2cs_handle, wdata, 256);
hal_i2c_master_receive_dma(&g_i2cm_handle, SLAVE_DEV_ADDR, rdata, 256);
while (hal_i2c_get_state(&g_i2cs_handle) ! = HAL_I2C_STATE_READY);
```

3.6.3.2 Test and Verification

1. Download *i2c_master_slave_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect I/O pins of the master to corresponding I/O pins of the slave.
3. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
4. You can view the data exchange results between I2C modules of the master and the slave in the **Receive Data** pane on GRUart.

3.7 I2S

The Inter-IC Sound Bus (I2S) refers to a bus standard introduced by Philips Semiconductor (now NXP Semiconductors) for audio data transmissions between digital audio devices. It is generally applicable to audio data transmissions for Bluetooth headsets and Bluetooth speakers.

The sections below focus on how to read/write data in interrupt, query, and DMA modes through the I2S interface and how to use the master/slave APIs.

3.7.1 I2S Master Audio

In the I2S Master Audio example in a GR551x SDK, the GR5515 SK Board is defined as the master to output example projects for I2S data streams. You need to configure parameters, including clock source, data size, and audio frequency, then call the initialization API to complete the configuration, and finally call the transmission API in interrupt and DMA modes to transfer data.

The source code and project file of the I2S Master Audio example are in `SDK_Folder\projects\peripheral\i2s\i2s_master_audio`, and project file is in the Keil_5 folder.

3.7.1.1 Code Interpretation

Figure 3-18 shows the workflow of the example project.

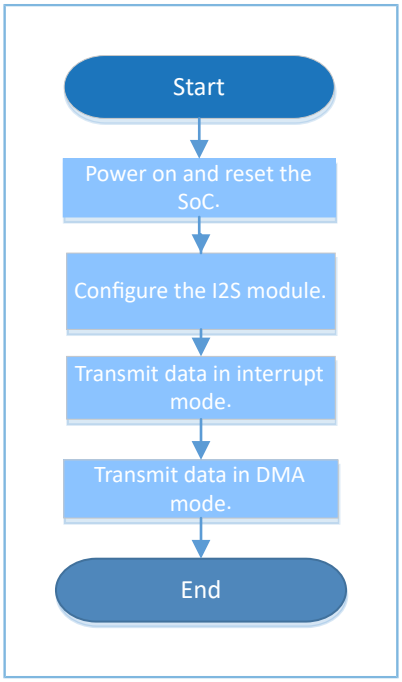

Figure 3-18 I2S Master Audio example workflow

1. Configure the I2S module.

   Configure the code data size as 32 bits, the clock source as 96 MHz, and the audio frequency as 48 kHz. Other values for these parameters are acceptable.

```
g_i2sm_handle.p_instance = I2S_M;
g_i2sm_handle.init.data_size = I2S_DATASIZE_32BIT;
g_i2sm_handle.init.clock_source = I2S_CLOCK_SRC_96M;
g_i2sm_handle.init.audio_freq = 48000;

hal_i2s_deinit(&g_i2sm_handle);
hal_i2s_init(&g_i2sm_handle);
```

   - init.data_size: I2S transfer data width. Options: I2S_DATASIZE_12BIT, I2S_DATASIZE_16BIT, I2S_DATASIZE_20BIT, I2S_DATASIZE_24BIT, and I2S_DATASIZE_32BIT.
   - init.clock_source: I2S clock source. Options: I2S_CLOCK_SRC_96M and I2S_CLOCK_SRC_32M.
   - init.audio_freq: audio frequency of the master. No settings are required on the slave.

   I2S pins are defined as follows. You can set other available pins according to Pin Mux tables in “Pin Mux” in *GR551x Datasheet*.

```
#define I2S_MASTER_WS_PIN                AON_GPIO_PIN_2
#define I2S_MASTER_TX_SDO_PIN           AON_GPIO_PIN_3
#define I2S_MASTER_RX_SDI_PIN           AON_GPIO_PIN_4
#define I2S_MASTER_SCLK_PIN             AON_GPIO_PIN_5
```

2. Call hal_i2s_transmit() to transfer data in polling mode. The code snippet is as follows:

```
hal_i2s_transmit(&g_i2sm_handle, wdata, sizeof(wdata) >> 2, 1000);
```

3. Call hal_i2s_transmit_it() to transfer data in DMA mode. Due to the non-blocking mode adopted, judge whether the transmission is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
hal_i2s_transmit_it(&g_i2sm_handle, wdata, sizeof(wdata) >> 2);
while (hal_i2s_get_state(&g_i2sm_handle) ! = HAL_I2S_STATE_READY);
```

4. Call hal_i2s_transmit_dma() to transfer data in DMA mode. Due to the non-blocking mode adopted, judge whether the transmission is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
hal_i2s_transmit_dma(&g_i2sm_handle, wdata, sizeof(wdata) >> 2);
while (hal_i2s_get_state(&g_i2sm_handle) ! = HAL_I2S_STATE_READY);
```

3.7.1.2 Test and Verification

1. Download *i2s_audio_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the data exchange results between I2S modules of the master and the slave in the **Receive Data** pane on GRUart.

3.7.2 I2S Master DMA UART

The I2S Master DMA UART example in a GR551x SDK verifies the P2P function of DMA. Data is sent to I2S from UART in DMA mode.

The source code and project file of the I2S Master DMA UART example are in `SDK_Folder\projects\peripheral\i2s\i2s_master_dma_uart`, and project file is in the Keil_5 folder.

3.7.2.1 Code Interpretation

Figure 3-19 shows the workflow of the example project.

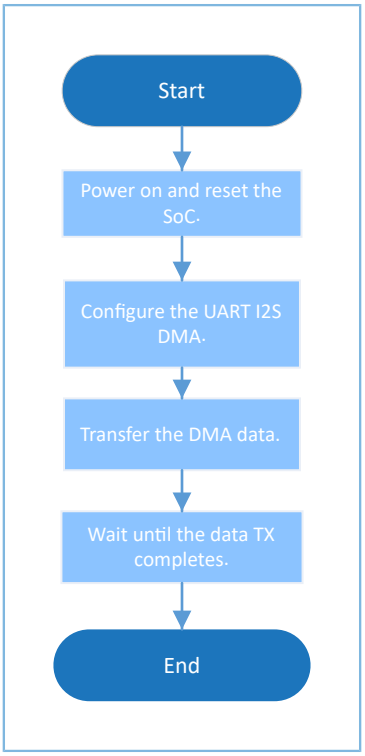

Figure 3-19 I2S Master DMA UART example workflow

1. Configure the I2S module.

```
hi2s->p_instance        = I2S_M;
hi2s->init.data_size    = I2S_DATASIZE_16BIT;
hi2s->init.clock_source = I2S_CLOCK_SRC_32M;
hi2s->init.audio_freq   = 4000;

hal_i2s_deinit(hi2s);
hal_i2s_init(hi2s);
```

   - For more information about I2S configurations, see "Section 3.7.1 I2S Master Audio". In this example project, init.data_size is configured as I2S_DATASIZE_16BIT, init.clock_source as I2S_CLOCK_SRC_32M, and init.audio_freq as 4000.
   - The UART is configured by calling app_log_init() and can be used as the debugging interface.

2. Enable I2S DMA, TX channel, and TX block, and refresh TX FIFO. The code snippet is as follows:

```
__HAL_I2S_ENABLE_DMA(hi2s);
/* Flush TX FIFO */
ll_i2s_clr_txfifo_all(hi2s->p_instance);
/* Enable channel TX */
ll_i2s_enable_tx(hi2s->p_instance, 0);
/* Enable TX block */
ll_i2s_enable_txblock(hi2s->p_instance);
```

3. Set the UART reception threshold and the DMA burst length. The code snippet is as follows:

```
ll_uart_set_rx_fifo_threshold(SERIAL_PORT_GRP, LL_UART_RX_FIFO_TH_QUARTER_FULL);
ll_dma_set_source_burst_length(DMA, hi2s->p_dmatx->channel, LL_DMA_SRC_BURST_LENGTH_8);
```

```
ll_dma_set_destination_burst_length(DMA, hi2s->p_dmatx->channel, LL_DMA_DST_BURST_LENGTH_8);
```

4. Wait until UART receives data. The code snippet is as follows:

```
/* Wait until receive any data */
while(!ll_uart_is_active_flag_rfne(SERIAL_PORT_GRP));
```

5. Start the DMA transfer, enable the TX FIFO empty interrupt, enable the I2S clock, and wait until the DMA transmission is completed. The code snippet is as follows:

```
hal_dma_start(hi2s->p_dmatx, (uint32_t)&UART0->RBR_DLL_THR, (uint32_t)&hi2s->p_instance-
>TXDMA, TEST_LENGTH);
__HAL_I2S_ENABLE_IT(hi2s, I2S_IT_TXFE);
/* Enable clock */
ll_i2s_enable_clock(hi2s->p_instance);
hal_dma_poll_for_transfer(hi2s->p_dmatx, 1000);
```

3.7.2.2 Test and Verification

1. Download *i2s_dma_uart_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the exchange results for data sent from UART to I2S in DMA mode in the **Receive Data** pane on GRUart.

3.7.3 I2S Master & Slave

The I2S Master & Slave example in a GR551x SDK verifies the data transfer function of I2S. The macro MASTER_BOARD is used to assign a master or a slave role.

This example requires two development boards; one (Board 1) serves as the master, and the other (Board 2) serves as the slave.

The source code and project file of the I2S Master & Slave example are in `SDK_Folder\projects\peripheral\i2s\i2s_master_salve\`, and project file is in the Keil_5 folder.

3.7.3.1 Code Interpretation

Figure 3-20 shows the workflow of the example project.

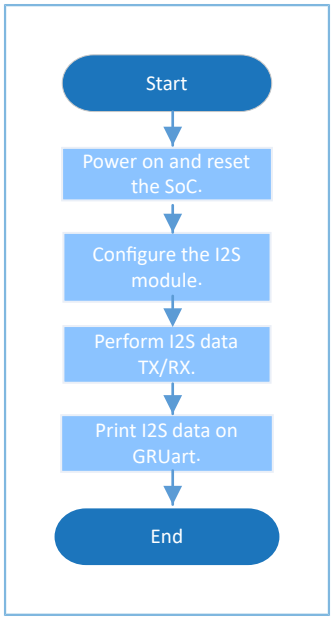

Figure 3-20 I2S Master & Slave example workflow

1. Configure the I2S module.

   For more information about I2S configurations, see "Section 3.7.1 I2S Master Audio".

```
#ifdef MASTER_BOARD
    g_i2s_handle.p_instance  = I2S_M;
    g_i2s_handle.init.audio_freq  = 48000;
#else
    g_i2s_handle.p_instance  = I2S_S;
#endif  /* MASTER_BOARD */
    g_i2s_handle.init.data_size    = I2S_DATASIZE_32BIT;
    g_i2s_handle.init.clock_source = I2S_CLOCK_SRC_96M;
    hal_i2s_init(&g_i2s_handle);
```

2. The master exchanges data with the slave in DMA/interrupt/polling mode.

   - Currently, the clock state for I2S is controlled by users. To prevent receiving or transferring non-target data, call i2s_flush_rx_fifo() or i2s_flush_tx_fifo() to clear data in RX/TX FIFO.
   - After operations complete, the master calls __HAL_I2S_DISABLE_CLOCK() to disable the clock or reinitialize I2S, so as to prevent the clock from enabling the device to remain in TX/RX state all the time.
   - Due to the non-blocking APIs adopted, judge whether the transmission/reception is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_receive_dma(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2);
while (!tx_rx_flag);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
sys_delay_ms(1);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);
hal_i2s_transmit_receive_it(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2);
while (!tx_rx_flag);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
```

```
sys_delay_ms(1);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_receive(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2, 2000);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
sys_delay_ms(1);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_it(&g_i2s_handle, wdata, sizeof(wdata) >> 2);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
sys_delay_ms(1);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);
sys_delay_ms(10);
i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_receive_it(&g_i2s_handle, rdata, sizeof(rdata) >> 2);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
sys_delay_ms(1);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);
```

3. The slave exchanges data with the master in DMA/interrupt/polling mode.

   To prevent receiving or transferring non-target data, call i2s_flush_rx_fifo() or i2s_flush_tx_fifo() to clear data in RX/TX FIFO. The code snippet is as follows:

```
i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_receive_dma(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2);
while (!tx_rx_flag);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_receive_it(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2);
while (!tx_rx_flag);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_receive(&g_i2s_handle, wdata, rdata, sizeof(wdata) >> 2, 2000);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_receive_it(&g_i2s_handle, rdata, sizeof(rdata) >> 2);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
hal_i2s_deinit(&g_i2s_handle);
hal_i2s_init(&g_i2s_handle);

i2s_flush_rx_fifo(&g_i2s_handle);
hal_i2s_transmit_it(&g_i2s_handle, wdata, sizeof(wdata) >> 2);
while (hal_i2s_get_state(&g_i2s_handle) ! = HAL_I2S_STATE_READY);
sys_delay_ms(2);
```

```
hal_i2s_deinit(&g_i2s_handle);
```

3.7.3.2 Test and Verification

1. Define the macro MASTER_BOARD in the project, compile and download it to Board 1; undefine the macro MASTER_BOARD in the project, compile and download it to Board 2.
2. Connect I/O pins of the master to corresponding I/O pins of the slave.
3. Connect the serial ports of both Board 1 and Board 2 to the PC respectively, start GRUart, and configure the serial ports on the GRUart as described in "Section 2.4 Serial Port Settings".
4. You can view the data transmission results for I2S in the **Receive Data** pane on GRUart.

3.8 PKC

The Public Key Cipher (PKC) module handles addition, subtraction, shift, and modulo operations for big integers exceeding 32 bits. It is generally used to generate calculation factors for encryption operations.

3.8.1 PKC

The PKC example in a GR551x SDK implements the modular addition, subtraction, multiplication, comparison, inversion, and shift operations for big integers, as well as addition, multiplication, RSA modular exponentiation, and ECC dot product operations for big integers.

In this section, an example of modular addition operation for big integers is provided, explaining the code, and test and verification details for a PKC example. For code about other operations, see the relevant example project code.

The source code and project file of the PKC example are in SDK_Folder\projects\peripheral\pkc\pkc, and project file is in the Keil_5 folder.

3.8.1.1 Code Interpretation

Figure 3-21 shows the workflow of a PKC example project (for modular addition operation for big integers).

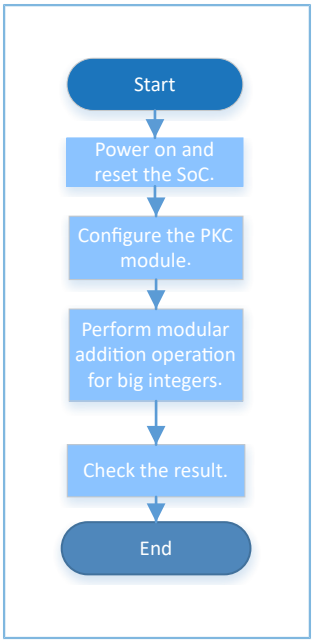

Figure 3-21 PKC example workflow

1. Configure the PKC module.

```
pkc_modular_add_t PKC_ModularAddStruct = {
    .p_A = In_a,
    .p_B = In_b,
    .p_P = In_n
};

PKCHandle.p_instance          = PKC;
PKCHandle.p_result            = Out_c;
PKCHandle.init.p_ecc_curve    = &ECC_CurveInitStruct;
PKCHandle.init.data_bits    = 256;
PKCHandle.init.secure_mode = PKC_SECURE_MODE_DISABLE;
PKCHandle.init.random_func = NULL;

hal_pkc_deinit(&PKCHandle);
hal_pkc_init(&PKCHandle);
```

- init.p_ecc_curve: pointer to the elliptic curve description
- init.data_bits: data bit
- init.secure_mode: security mode. Options: PKC_SECURE_MODE_DISABLE and PKC_SECURE_MODE_ENABLE
- init.random_func: pointer to a random number generation function

2. Call the blocking API hal_pkc_modular_add() or the interrupt API hal_pkc_modular_add_it() to perform a modular addition operation on the big integers In_a and In_b. The results are output to Out_c and compared with Exc_c. The code snippet is as follows:

```
if (HAL_OK != hal_pkc_modular_add(&PKCHandle, &PKC_ModularAddStruct, 1000))
{
     printf("\r\n%dth add operation got error\r\n", count++);
     continue;
}
```

```
if (check_result((uint8_t*)"pkc_modular_add", Out_c, Exc_c, PKCHandle.init.data_bits))
{
     error++;
}

if (HAL_OK != hal_pkc_modular_add_it(&PKCHandle, &PKC_ModularAddStruct))
{
     printf("\r\n%dth add it operation got error\r\n", count++);
     continue;
}
while(int_done_flag == 0);
if (check_result((uint8_t*)"pkc_modular_add_it", Out_c, Exc_c, PKCHandle.init.data_bits))
{
     error++;
}
```

3.8.1.2 Test and Verification

1. Download *pkc_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the data operation results for PKC in the **Receive Data** pane on GRUart.

3.9 PWM

The Pulse Width Modulation (PWM) is a method to perform digital coding for analog signal level. It is generally used to output square waves or control LEDs to flash at fixed frequencies.

The sections below focus on outputting signals with controllable frequency and duty ratio in flicker/breath mode by using PWM to control LED states.

3.9.1 PWM Breath

The PWM Breath example in a GR551x SDK implements the breath mode of PWM. When you run the example project, you can use a logic analyzer to observe the changes in the duty ratio of the waveform from 0% to 100%. You can also observe the changes by flash frequency of the LEDs.

The source code and project file of the PWM Breath example are in SDK_Folder\projects\peripheral\pwm\pwm_breath, and project file is in the Keil_5 folder.

3.9.1.1 Code Interpretation

Figure 3-22 shows the workflow of the example project.

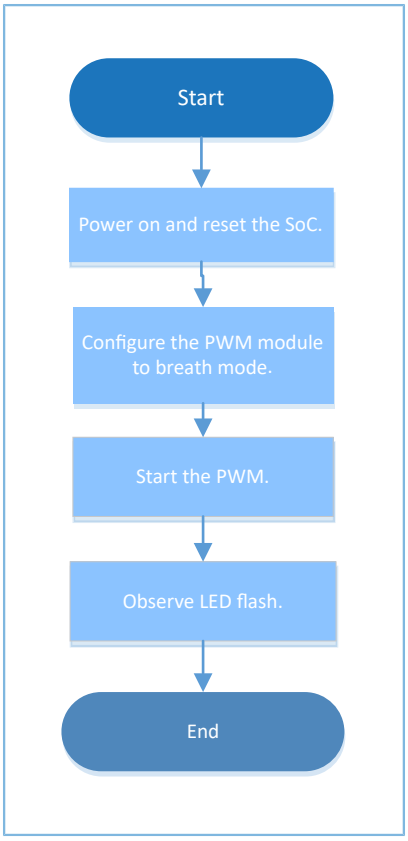

Figure 3-22 PWM Breath example workflow

1. Configure the PWM module.

```
led1_handle.p_instance = PWM1_MODULE;
led1_handle.active_channel = HAL_PWM_ACTIVE_CHANNEL_B;
led1_handle.init.mode = PWM_MODE_BREATH;
led1_handle.init.align = PWM_ALIGNED_EDGE;
led1_handle.init.freq = 100;   //PWM output freq = 100Hz
led1_handle.init.bperiod = 500;  //breath period = 500ms
led1_handle.init.hperiod = 200;  //hold period = 200ms, max = 262ms
led1_handle.init.channel_b.duty = 50;
led1_handle.init.channel_b.drive_polarity = PWM_DRIVEPOLARITY_POSITIVE;
hal_pwm_init(&led1_handle);

led2_handle.p_instance = PWM0_MODULE;
led2_handle.active_channel = HAL_PWM_ACTIVE_CHANNEL_C;
led2_handle.init.mode = PWM_MODE_BREATH;
led2_handle.init.align = PWM_ALIGNED_EDGE;
led2_handle.init.freq = 100;  //PWM output freq = 100Hz
led2_handle.init.bperiod = 500; //breath period = 500ms
led2_handle.init.hperiod = 200; //hold period = 200ms, max = 262ms
led2_handle.init.channel_c.duty = 50;
led2_handle.init.channel_c.drive_polarity = PWM_DRIVEPOLARITY_POSITIVE;
hal_pwm_init(&led2_handle);
```

- active_channel: PWM channel. Options: HAL_PWM_ACTIVE_CHANNEL_A, HAL_PWM_ACTIVE_CHANNEL_B, HAL_PWM_ACTIVE_CHANNEL_C, HAL_PWM_ACTIVE_CHANNEL_ALL, and HAL_PWM_ACTIVE_CHANNEL_CLEARED

- init.mode: PWM mode. Options: PWM_MODE_FLICKER and PWM_MODE_BREATH
- init.align: PWM alignment mode. Options: PWM_ALIGNED_EDGE and PWM_ ALIGNED_CENTER. The alignment mode cannot be configured when in PWM breath mode.
- init.freq: PWM frequency. Range: 0 to SystemCoreClock/2
- init.bperiod: specified PWM breath period in breath mode. Range: 0 to 0xFFFFFFFF/(SystemCoreClock x 1000) (ms)
- init.hperiod: specified PWM hold period in breath mode. Range: 0 to 0xFFFFFFFF/(SystemCoreClock x 1000) (ms)
- init.channel_b.duty: duty ratio of PWM B channel in output mode. Range: 0 to 100. The duty ratio cannot be configured when in PWM breath mode.
- init.channel_b.drive_polarity: driver polarity of PWM B channel in output mode. Options: PWM_DRIVEPOLARITY_NEGATIVE and PWM_DRIVEPOLARITY_POSITIVE

2. Call led_breath_on() to enable the PWM breath mode. The code snippet is as follows:

```
void led_breath_on(uint8_t id)
{
    if (id == LED1)
    {
        hal_pwm_start(&led1_handle);
    }
    else if (id == LED2)
    {
        hal_pwm_start(&led2_handle);
    }
}
```

3.9.1.2 Test and Verification

1. Download *pwm_breath_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the PWM implementation process in the **Receive Data** pane on GRUart.
4. Use a logic analyzer to observe the changes in the duty ratio of the waveform from 0% to 100%.

3.9.2 PWM Flicker

The PWM Flicker example in a GR551x SDK implements the fixed duty ratio mode of PWM. When you run the example project, you can use a logic analyzer to observe whether the waveform has a fixed duty ratio, or observe whether the LED flashes at a fixed frequency.

The source code and project file of the PWM Flicker example are in SDK_Folder\projects\peripheral\pwm\pwm_flicker, and project file is in the Keil_5 folder.

3.9.2.1 Code Interpretation

Figure 3-23 shows the workflow of the example project.

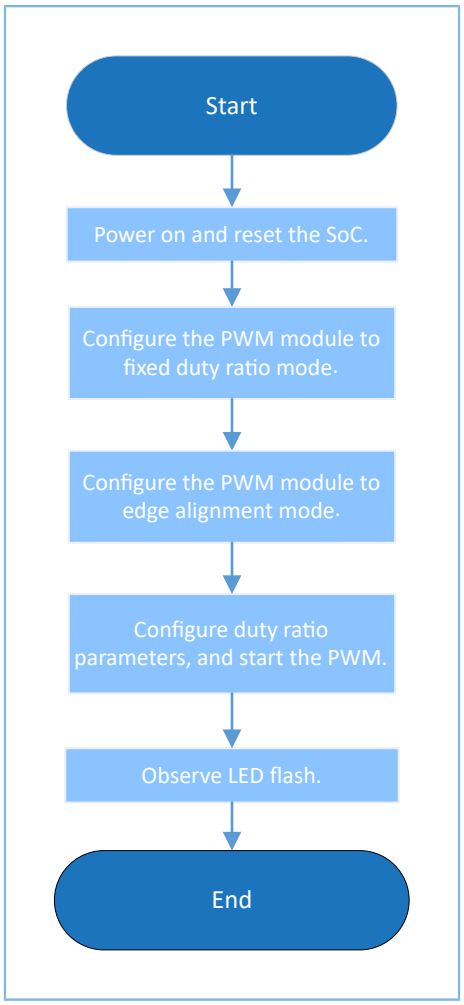

Figure 3-23 PWM Flicker example workflow

1. Configure the PWM module.

```
led1_handle.p_instance = PWM1_MODULE;
led1_handle.active_channel = HAL_PWM_ACTIVE_CHANNEL_B;
led1_handle.init.mode = PWM_MODE_FLICKER;
led1_handle.init.align = PWM_ALIGNED_EDGE;
led1_handle.init.freq = 1000;           //PWM output freq = 100Hz
led1_handle.init.channel_b.duty = 50;
led1_handle.init.channel_b.drive_polarity = PWM_DRIVEPOLARITY_NEGATIVE;
hal_pwm_init(&led1_handle);

led2_handle.p_instance = PWM0_MODULE;
led2_handle.active_channel = HAL_PWM_ACTIVE_CHANNEL_C;
led2_handle.init.mode = PWM_MODE_FLICKER;
led2_handle.init.align = PWM_ALIGNED_EDGE;
led2_handle.init.freq = 1000;           //PWM output freq = 100Hz
led2_handle.init.channel_c.duty = 50;
led2_handle.init.channel_c.drive_polarity = PWM_DRIVEPOLARITY_NEGATIVE;
```

```
hal_pwm_init(&led2_handle);
```

For more information about PWM configurations, see "Section 3.9.1 PWM Breath". In this example, set init.mode to PWM_MODE_FLICKER.

2. Call led_light() to enable the PWM flicker mode. The code snippet is as follows:

```
void led_light(uint8_t id, uint8_t light)
{
    pwm_channel_init_t pwm_channel;

    pwm_channel.drive_polarity = PWM_DRIVEPOLARITY_NEGATIVE;
    pwm_channel.duty = (uint16_t)light * 100/255;
    if (id == LED1)
    {
        hal_pwm_stop(&led1_handle);
        hal_pwm_config_channel(&led1_handle, &pwm_channel, HAL_PWM_ACTIVE_CHANNEL_B);
        hal_pwm_start(&led1_handle);
    }
    else if (id == LED2)
    {
        hal_pwm_stop(&led2_handle);
        hal_pwm_config_channel(&led2_handle, &pwm_channel, HAL_PWM_ACTIVE_CHANNEL_C);
        hal_pwm_start(&led2_handle);
    }
}
```

3.9.2.2 Test and Verification

1. Download *pwm_flicker_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the PWM implementation process in the **Receive Data** pane on GRUart.
4. Use a logic analyzer to observe whether the waveform has a fixed duty ratio. The open channel is aligned to the start of the duty cycle in PWM edge alignment mode, and aligned around the middle point of the duty cycle (as the axis) symmetrically in PWM center alignment mode. Observe whether the LED flashes at a fixed frequency.

3.10 RNG

The Random Number Generator (RNG) refers to a program or hardware to generate random numbers. It is generally used to generate random numbers for other applications.

The sections below focus on how to generate random numbers in interrupt/query mode on a GR551x SoC.

3.10.1 RNG Interrupt

The RNG Interrupt example in a GR551x SDK shows how to get random numbers in interrupt mode. In this example, seeds provided by users and by switching oscillator S0 are used to generate random numbers.

The source code and project file of the RNG Interrupt example are in `SDK_Folder\projects\peripheral\rng\rng_interrupt`, and project file is in the Keil_5 folder.

3.10.1.1 Code Interpretation

Figure 3-24 shows the workflow of the example project.

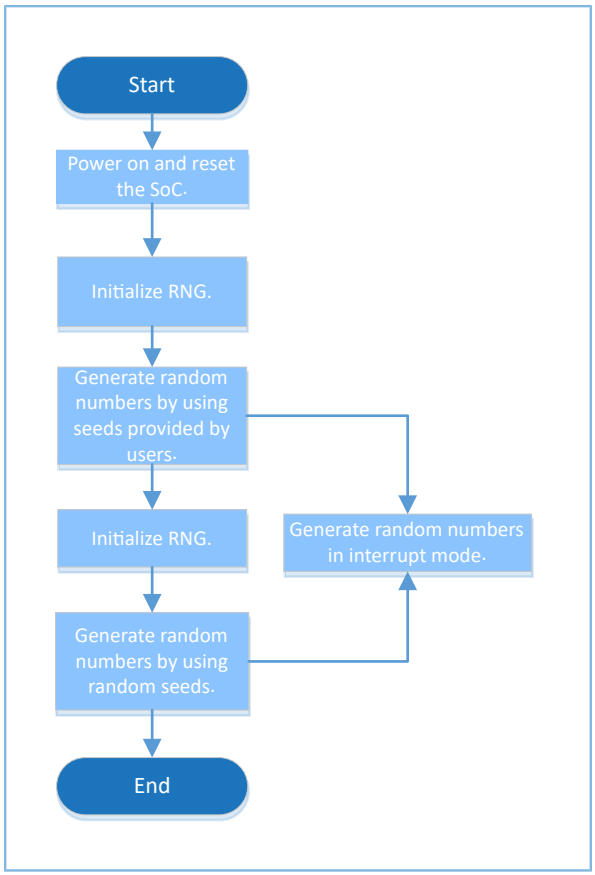

Figure 3-24 RNG Interrupt example workflow

1. Initialize random numbers generated by using seeds provided by users.

```
uint16_t g_random_seed[8] = {0x1234, 0x5678, 0x90AB, 0xCDEF, 0x1468, 0x2345, 0x5329,
 0x2411};

g_rng_handle.init.seed_mode  = RNG_SEED_USER;
g_rng_handle.init.lfsr_mode  = RNG_LFSR_MODE_59BIT;
g_rng_handle.init.out_mode   = RNG_OUTPUT_LFSR;
g_rng_handle.init.post_mode  = RNG_POST_PRO_NOT;

hal_rng_deinit(&g_rng_handle);
hal_rng_init(&g_rng_handle);
```

- seed_mode: seed source. Options: RNG_SEED_FR0_S0 and RNG_SEED_USER
- lfsr_mode: LFSR mode. Options: RNG_LFSR_MODE_59BIT and RNG_LFSR_MODE_128BIT
- out_mode: output mode. Options: RNG_OUTPUT_FR0_S0, RNG_OUTPUT_CYCLIC_PARITY, and RNG_OUTPUT_CYCLIC, and RNG_OUTPUT_LFSR
- post_mode: post-processing mode. Options: RNG_POST_PRO_NOT, RNG_POST_PRO_SKIPPING, RNG_POST_PRO_COUNTING, and RNG_POST_PRO_NEUMANN

Note:

When seed_mode is configured as RNG_SEED_USER, out_mode cannot be configured as RNG_OUTPUT_FR0_S0.

2. Initialize random numbers generated by using seeds provided by switching oscillator S0.

```
g_rng_handle.init.seed_mode  = RNG_SEED_FR0_S0;
g_rng_handle.init.lfsr_mode  = RNG_LFSR_MODE_128BIT;
g_rng_handle.init.out_mode   = RNG_OUTPUT_FR0_S0;

hal_rng_deinit(&g_rng_handle);
hal_rng_init(&g_rng_handle);
```

- seed_mode: seed source. Select RNG_SEED_FR0_S0 (a seed provided by switching oscillator S0).
- lfsr_mode: LFSR mode. Select RNG_LFSR_MODE_128BIT.
- out_mode: output mode. Select RNG_OUTPUT_FR0_S0.

3. Generate random numbers by using interrupt APIs. The code snippet is as follows:

```
hal_rng_generate_random_number_it(&g_rng_handle, g_random_seed);
```

3.10.1.2 Test and Verification

1. Download *rng_interrupt_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the generated random numbers in the **Receive Data** pane on GRUart.

3.10.2 RNG Query

The RNG Query example in a GR551x SDK shows how to get random numbers in blocking mode. In this example, seeds provided by users and by switching oscillator S0 are used to generate random numbers.

The source code and project file of the RNG Query example are in SDK_Folder\projects\peripheral\rng\rng_query, and project file is in the Keil_5 folder.

3.10.2.1 Code Interpretation

Figure 3-25 shows the workflow of the example project.

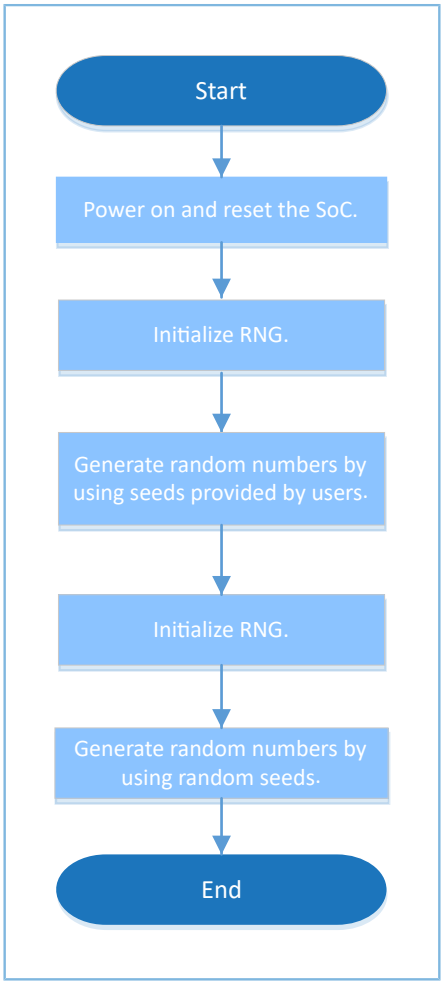

Figure 3-25 RNG Query example workflow

1. Initialize random numbers generated by using seeds provided by users.

```
uint16_t g_random_seed[8] = {0x1234, 0x5678, 0x90AB, 0xCDEF, 0x1468, 0x2345, 0x5329,
 0x2411};

g_rng_handle.init.seed_mode  = RNG_SEED_USER;
g_rng_handle.init.lfsr_mode  = RNG_LFSR_MODE_59BIT;
g_rng_handle.init.out_mode   = RNG_OUTPUT_LFSR;
g_rng_handle.init.post_mode  = RNG_POST_PRO_NOT;

hal_rng_deinit(&g_rng_handle);
hal_rng_init(&g_rng_handle);
```

For more information about RNG configurations, see "Section 3.10.1 RNG Interrupt".

2. Initialize random numbers generated by using seeds provided by switching oscillator S0.

```
g_rng_handle.init.seed_mode  = RNG_SEED_FR0_S0;
g_rng_handle.init.lfsr_mode  = RNG_LFSR_MODE_128BIT;
g_rng_handle.init.out_mode   = RNG_OUTPUT_FR0_S0;

hal_rng_deinit(&g_rng_handle);
```

```
hal_rng_init(&g_rng_handle);
```

For more information about RNG configurations, see “Section 3.10.1 RNG Interrupt”.

3. Generate random numbers by using APIs in blocking mode.

```
hal_rng_generate_random_number(&g_rng_handle, g_random_seed, &data);
```

3.10.2.2 Test and Verification

1. Download *rng_query_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the generated random numbers in the **Receive Data** pane on GRUart.

3.11 RTC

The Real-Time Clock (RTC) provides not only precise real time, but also precise time base for electronic systems. It is generally used for timing or for alarm applications.

The sections below focus on implementing the Calendar function by using the RTC.

3.11.1 Calendar

The Calendar example in a GR551x SDK implements the calendar and alarm functions. The current time is printed every second in the Receive Data pane on GRUart.

The source code and project file of the Calendar example are in `SDK_Folder\projects\peripheral\rtc\calendar`, and project file is in the Keil_5 folder.

3.11.1.1 Code Interpretation

Figure 3-26 shows the workflow of the example project.

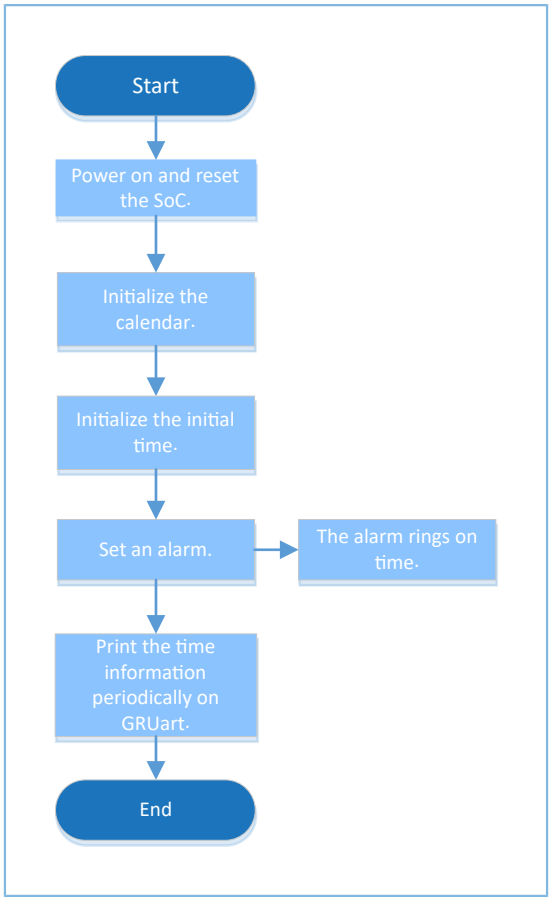

Figure 3-26 Calendar example workflow

1. Initialize the calendar.

```
calendar_time_t time;
calendar_alarm_t alarm;
g_systick_flag = 0;

hal_calendar_init(&g_calendar_handle);
```

- alarm_sel: alarm type. Options: CALENDAR_ALARM_SEL_DATE and CALENDAR_ALARM_SEL_WEEKDAY
- alarm_date_week_mask: the mask generated by the alarm clock. When CALENDAR_ALARM_SEL_WEEKDAY is set, this parameter can be one of or any combination of CALENDAR_ALARM_WEEKDAY_SUN to CALENDAR_ALARM_WEEKDAY_SAT; when CALENDAR_ALARM_SEL_DATE is set, this parameter ranges from 1 to 31.
- hour: the hour of the alarm time
- min: the minute of the alarm time

2. Set the initial date as 2019-05-20 08:00. The code snippet is as follows:

```
time.year = 19;
time.mon  = 5;
time.date = 20;
time.hour = 8;
```

```
time.min  = 0;
time.sec  = 0;
hal_calendar_init_time(&g_calendar_handle, &time)
```

3. Set an alarm at 08:01 for each weekday. This function shall be called each time after the system time is set. The code snippet is as follows:

```
alarm.alarm_sel = CALENDAR_ALARM_SEL_WEEKDAY;
alarm.alarm_date_week_mask = 0x3E;
alarm.hour = 8;
alarm.min  = 1;
hal_calendar_set_alarm(&g_calendar_handle, &alarm);
```

4. Get the current time. The code snippet is as follows:

```
hal_calendar_get_time(&g_calendar_handle, &time);
```

3.11.1.2 Test and Verification

1. Download *calendar_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. The current time is printed every second in the **Receive Data** pane on GRUart.

3.11.2 Alarm

The Alarm example in a GR551x SDK implements the multi-alarm function. The current time is printed every second in the Receive Data pane on GRUart.

The source code and project file of the Alarm example are in SDK_Folder\projects\peripheral\rtc\alarm, and project file is in the Keil_5 folder.

3.11.2.1 Code Interpretation

Figure 3-27 shows the workflow of the example project.

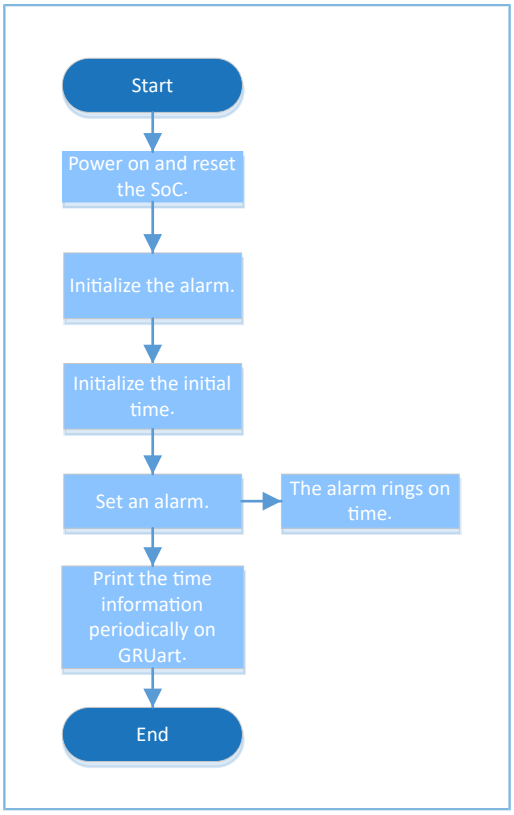

Figure 3-27 Alarm example workflow

1. Initialize the alarm.

```
app_time_t s_time = {0, 8, 9, 28, 10, 19 , 0, 0};
app_alarm_t s_alarm;
app_drv_err_t app_err_code;

app_err_code = app_alarm_init(alarm_data_tag, app_alarm_fun);
if (app_err_code ! = APP_DRV_SUCCESS)
{
    printf(" Initializing the alarm failed!\r\n");
    return;
}
app_err_code = app_alarm_set_time(&s_time);
if (app_err_code ! = APP_DRV_SUCCESS)
{
    printf(" Initializing the alarm time failed!\r\n");
    return;
}

app_alarm_reload();
app_alarm_del_all();

s_alarm.hour = 9;
s_alarm.min  = 10;
s_alarm.alarm_sel = CALENDAR_ALARM_SEL_WEEKDAY;
s_alarm.alarm_date_week_mask = 0x7F;
app_alarm_add(&s_alarm, alarm_0);
```

For more information about ALARM configurations, see “Section 3.11.1 Calendar”.

2. Initialize the alarm and set an alarm user callback function. The code snippet is as follows:

```
app_alarm_init(alarm_data_tag, app_alarm_fun);
```

3. Set the system time. The code snippet is as follows:

```
app_alarm_set_time(&s_time);
```

4. Reload the system alarms. This function shall be called each time after the system time is set. The code snippet is as follows:

```
app_alarm_reload();
```

5. Delete all the existing alarms. The code snippet is as follows:

```
app_alarm_del_all();
```

6. Add an alarm for the system. The code snippet is as follows:

```
app_alarm_add(&s_alarm, alarm_0);
```

3.11.2.2 Test and Verification

1. Download *alarm_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. The current time is printed every second in the **Receive Data** pane on GRUart.

3.12 SPI

The Serial Peripheral Interface (SPI) is a type of high-speed full-duplex synchronous communication bus. It is generally used to connect external sensors or flash memories.

The sections below focus on how to read/write data in interrupt, query, and DMA modes through SPI and how to use the master/slave APIs.

3.12.1 SPIM DMA

The SPIM DMA example in a GR551x SDK shows how to transfer data through SPI master (SPIM) in DMA mode.

The source code and project file of the SPIM DMA example are in SDK_Folder\projects\peripheral\spi\spi_master_dma, and project file is in the Keil_5 folder.

3.12.1.1 Code Interpretation

Figure 3-28 shows the workflow of the example project.

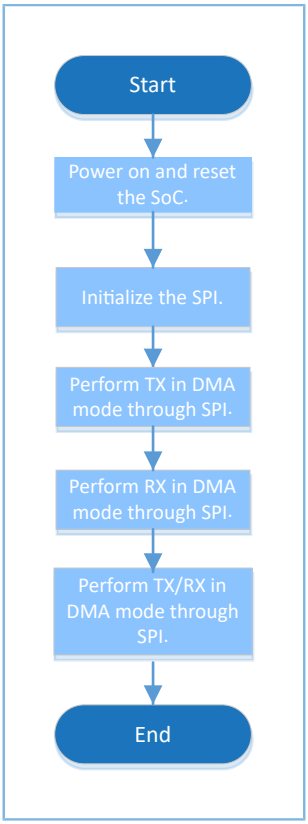

Figure 3-28 SPIM DMA example workflow

1. Initialize the SPIM DMA.

```
g_spim_handle.init.data_size    = SPI_DATASIZE_8BIT;
g_spim_handle.init.clock_polarity   = SPI_POLARITY_LOW;
g_spim_handle.init.clock_phase    = SPI_PHASE_1EDGE;
g_spim_handle.init.baudrate_prescaler = SystemCoreClock/2000000;
g_spim_handle.init.ti_mode    = SPI_TIMODE_DISABLE;
g_spim_handle.init.slave_select    = SPI_SLAVE_SELECT_0;
hal_spi_init(&g_spim_handle);
```

- data_size: SPI transfer data width. Options: SPI_DATASIZE_4BIT to SPI_DATASIZE_32BIT
- clock_polarity: clock signal in idle state. Options: SPI_POLARITY_LOW and SPI_POLARITY_HIGH
- clock_phase: clock switching time. Options: SPI_PHASE_1EDGE and SPI_PHASE_2EDGE
- baudrate_prescaler: SPI clock. It is set to SystemCoreClock/2000000 (20 MHz) in this example.
- ti_mode: Enable/Disable the TI mode. Options: SPI_TIMODE_DISABLE and SPI_TIMODE_ENABLE
- slave_select: slave. Options: SPI_SLAVE_SELECT_0, SPI_SLAVE_SELECT_1, and SPI_SLAVE_SELECT_ALL

2. TX API in DMA mode through SPI. The code snippet is as follows:

```
hal_spi_transmit_dma(&g_spim_handle, tx_buffer, sizeof(tx_buffer));
```

3. RX API in DMA mode through SPI. The code snippet is as follows:

```
hal_spi_receive_dma(&g_spim_handle, rx_buffer, sizeof(rx_buffer));
```

4. TX/RX API in DMA mode through SPI. The code snippet is as follows:

```
hal_spi_transmit_receive_dma(&g_spim_handle, tx_buffer, rx_buffer, sizeof(rx_buffer));
```

3.12.1.2 Test and Verification

1. Download *spim_dma_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the data transmission results for SPI in the **Receive Data** pane on GRUart.

3.12.2 SPIM DMA UART

The SPIM DMA UART example in a GR551x SDK shows how to transfer data from UART to SPI in DMA mode and then send the data in DMA mode through SPI. In this example, a logic analyzer is used to check data accuracy.

The source code and project file of the SPIM DMA UART example are in SDK_Folder\projects\peripheral\spi\spim_dma_uart, and project file is in the Keil_5 folder.

3.12.2.1 Code Interpretation

Figure 3-29 shows the workflow of the example project.

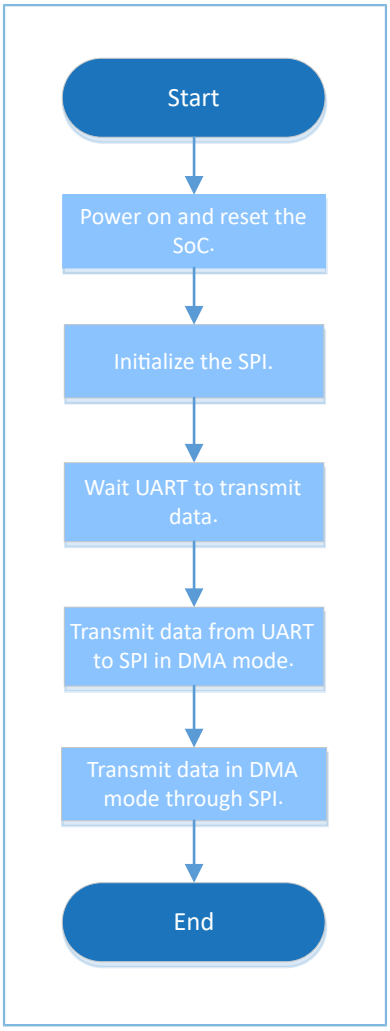

Figure 3-29 SPIM DMA UART example workflow

1. Initialize the SPIM DMA UART.

```
g_spim_handle.init.data_size    = SPI_DATASIZE_32BIT;
g_spim_handle.init.clock_polarity    = SPI_POLARITY_LOW;
g_spim_handle.init.clock_phase    = SPI_PHASE_1EDGE;
g_spim_handle.init.baudrate_prescaler = SystemCoreClock/2000000;
g_spim_handle.init.ti_mode      = SPI_TIMODE_DISABLE;
g_spim_handle.init.slave_select     = SPI_SLAVE_SELECT_0;
hal_spi_init(&g_spim_handle);
```

For more information about SPI configurations, see “Section 3.12.1 SPIM DMA”.

2. Configure the SPI in write mode. The code snippet is as follows:

```
__HAL_SPI_DISABLE(hspi);
ll_spi_set_transfer_direction(hspi->p_instance, LL_SSI_SIMPLEX_TX);
__HAL_SPI_ENABLE(hspi);
```

3. Set the UART reception threshold and the DMA burst length. The code snippet is as follows:

```
ll_uart_set_rx_fifo_threshold(SERIAL_PORT_GRP, LL_UART_RX_FIFO_TH_CHAR_1);
```

```
ll_dma_set_source_burst_length(DMA, hspi->p_dmatx->channel, LL_DMA_SRC_BURST_LENGTH_1);
ll_dma_set_destination_burst_length(DMA, hspi->p_dmatx->channel, LL_DMA_DST_BURST_LENGTH_4);
```

4. Wait until UART receives data. The code snippet is as follows:

```
while(!ll_uart_is_active_flag_rfne(SERIAL_PORT_GRP));
```

5. Start DMA transmission, and wait until the transmission is completed. The code snippet is as follows:

```
hal_dma_start(hspi->p_dmatx, (uint32_t)&UART0->RBR_DLL_THR,
              (uint32_t)&hspi->p_instance->DATA, TEST_LENGTH);
__HAL_SPI_ENABLE_DMATX(hspi);
hal_dma_poll_for_transfer(hspi->p_dmatx, 1000);
```

3.12.2.2 Test and Verification

1. Download *spim_dma_uart_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the data transmission results for SPI in the **Receive Data** pane on GRUart.
4. Use a logic analyzer to check the data accuracy.

3.12.3 SPIM ADXL345

The SPIM ADXL345 example in a GR551x SDK shows how to read the acceleration information from ADXL345 through SPI and print the information on GRUart.

The source code and project file of the SPIM ADXL345 example are in SDK_Folder\projects\peripheral\spi\spi_master_adxl345, and project file is in the Keil_5 folder.

3.12.3.1 Code Interpretation

Figure 3-30 shows the workflow of the example project.

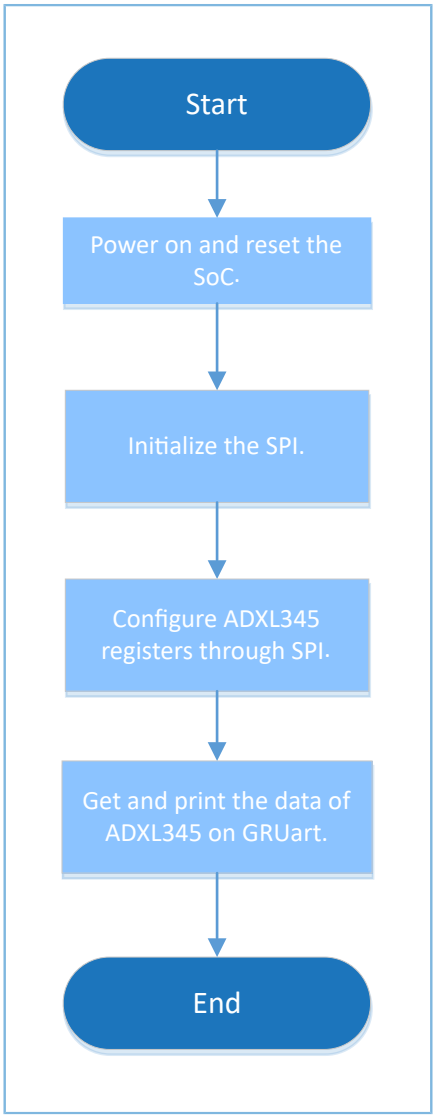

Figure 3-30 SPIM ADXL345 example workflow

1. Initialize the SPIM ADXL345. The code snippet is as follows:

```
g_spim_handle.init.data_size  = SPI_DATASIZE_8BIT;
g_spim_handle.init.clock_polarity = SPI_POLARITY_HIGH;
g_spim_handle.init.clock_phase  = SPI_PHASE_2EDGE;
g_spim_handle.init.baudrate_prescaler = SystemCoreClock/2000000;
g_spim_handle.init.ti_mode   = SPI_TIMODE_DISABLE;
g_spim_handle.init.slave_select = SPI_SLAVE_SELECT_0;

hal_spi_init(&g_spim_handle);
```

   For more information about SPI configurations, see “Section 3.12.1 SPIM DMA”.

2. Read data from ADXL345 by using hal_spi_read_eeprom(). The code snippet is as follows:

```
wdata[0] = read_cmd + 0x0;
hal_spi_read_eeprom(&g_spim_handle, wdata, rdata, 1, 1, 5000);
```

3. Transmit data to ADXL345 by using hal_spi_transmit(). The code snippet is as follows:

```
/* normal, 100Hz */
wdata[0] = write_cmd + 0x2C;
wdata[1] = 0x0A;
wdata[2] = 0x08;
wdata[3] = 0x00;
hal_spi_transmit(&g_spim_handle, wdata, 4, 5000);

/* full-bits, 4mg/LSB */
wdata[0] = write_cmd + 0x31;
wdata[1] = 0x09;
hal_spi_transmit(&g_spim_handle, wdata, 2, 5000);

/* FIFO flow */
wdata[0] = write_cmd + 0x38;
wdata[1] = 0x80;
hal_spi_transmit(&g_spim_handle, wdata, 2, 5000);
```

4. Read data from ADXL345 by using hal_spi_read_eeprom().

```
wdata[0] = read_cmd + 0x1D;
hal_spi_read_eeprom(&g_spim_handle, wdata, rdata, 1, 29, 5000);

wdata[0] = read_cmd + 0x32;
hal_spi_read_eeprom(&g_spim_handle, wdata, rdata, 1, 6, 5000);
```

3.12.3.2 Test and Verification

1. Download *spim_adxl345_sk_r2_fw.bin* to the GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the data from the ADXL345 sensor in the **Receive Data** pane on GRUart.

3.12.4 SPI Master & Slave

The SPI Master & Slave example in a GR551x SDK implements data exchange between the SPI master and the SPI slave. Code for both the master and the slave is implemented in this example. To use this example, you need to define the master and the slave by using the macro MASTER_BOARD.

This example requires two development boards; one (Board 1) serves as the master, and the other (Board 2) serves as the slave.

The source code and project file of the SPI Master & Slave example are in SDK_Folder\projects\peripheral\spi\spi_master_slave, and project file is in the Keil_5 folder.

3.12.4.1 Code Interpretation

Figure 3-31 shows the workflow of the example project.

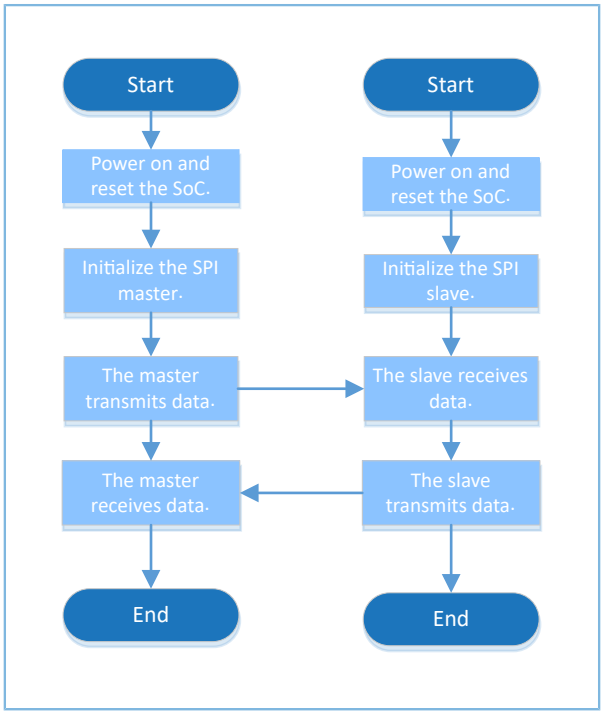

Figure 3-31 SPI Master & Slave example workflow

1. Initialize the master of the SPI Master & Slave.

```
g_spi_handle.p_instance  = SPIM;
g_spi_handle.init.data_size  = SPI_DATASIZE_32BIT;
g_spi_handle.init.clock_polarity  = SPI_POLARITY_LOW;
g_spi_handle.init.clock_phase = SPI_PHASE_1EDGE;
g_spi_handle.init.baudrate_prescaler = SystemCoreClock/1000000;
g_spi_handle.init.ti_mode   = SPI_TIMODE_DISABLE;
g_spi_handle.init.slave_select  = SPI_SLAVE_SELECT_0;
hal_spi_init(&g_spim_handle);
```

For more information about SPI configurations, see "Section 3.12.1 SPIM DMA".

2. Initialize the slave of the SPI Master & Slave.

```
g_spi_handle.p_instance  = SPIS;
g_spi_handle.init.data_size  = SPI_DATASIZE_32BIT;
g_spi_handle.init.clock_polarity = SPI_POLARITY_LOW;
g_spi_handle.init.clock_phase = SPI_PHASE_1EDGE;
g_spi_handle.init.ti_mode  = SPI_TIMODE_DISABLE;
hal_spi_init(&g_spim_handle);
```

For more information about SPI configurations, see "Section 3.12.1 SPIM DMA".

3. The SPI master transmits or receives data in interrupt mode. Due to the non-blocking APIs adopted, judge whether the transmission/reception is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
g_tx_done = 0;
hal_spi_transmit_it(&g_spi_handle, wdata, sizeof(wdata));
/* Wait for send done */
while(!g_tx_done);
```

```
g_rx_done = 0;
hal_spi_receive_it(&g_spi_handle, rdata, sizeof(rdata));
/* Wait for receive done */
while(!g_rx_done);
```

4. The SPI slave receives or transmits data in interrupt mode. Due to the non-blocking APIs adopted, judge whether the transmission/reception is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
hal_spi_receive_it(&g_spi_handle, rdata, sizeof(rdata))
/* Wait for receive done */
while(!g_rx_done);

g_tx_done = 0;
hal_spi_transmit_it(&g_spi_handle, wdata, sizeof(wdata))
while(!g_tx_done);
```

5. The master and the slave transmits/receives data by using non-blocking APIs, and results or states are returned by hal_spi_rx_cplt_callback(), hal_spi_tx_cplt_callback(), hal_spi_tx_rx_cplt_callback(), or hal_spi_abort_cplt_callback(). Users can customize executable operations in these four functions. The code snippet is as follows:

```
void hal_spi_rx_cplt_callback(spi_handle_t *hspi)
{
    g_rx_done = 1;
}

void hal_spi_tx_cplt_callback(spi_handle_t *hspi)
{
    g_tx_done = 1;
}

void hal_spi_tx_rx_cplt_callback(spi_handle_t *p_spi)
{
    g_rx_done = 1;
    g_tx_done = 1;
}

void hal_spi_abort_cplt_callback(spi_handle_t *hspi)
{
    if (hspi->p_instance == SPIM)
        printf("This is Abort complete Callback in SPIM.\r\n");
    else if (hspi->p_instance == SPIS)
        printf("This is Abort complete Callback in SPIS.\r\n");
}
```

3.12.4.2 Test and Verification

1. Define the macro MASTER_BOARD in the project, compile and download it to Board 1; undefine the macro MASTER_BOARD in the project, compile and download it to Board 2.
2. Connect I/O pins of the master to corresponding I/O pins of the slave.

3. Connect the serial ports of both Board 1 and Board 2 to the PC respectively, start GRUart, and configure the serial ports on the GRUart as described in "Section 2.4 Serial Port Settings".
4. You can view the data exchange results between the SPI master and the SPI slave in the **Receive Data** pane on GRUart.

3.13 UART

Universal Asynchronous Receiver/Transmitter (UART) can operate in full-duplex mode (in which both devices transmit and receive at the same time). The UART example is generally used to debug programs or to exchange data with other peripherals.

This section focuses on data transfer of UART in DMA, interrupt, and blocking modes.

3.13.1 UART DMA

The UART DMA example project enables a UART to receive and transmit data in DMA mode.

The source code and project file of the UART DMA example are in `SDK_Folder\projects\peripheral\uart\uart_dma`, and project file is in the Keil_5 folder.

3.13.1.1 Code Interpretation

Figure 3-32 shows the workflow of a UART DMA example project.

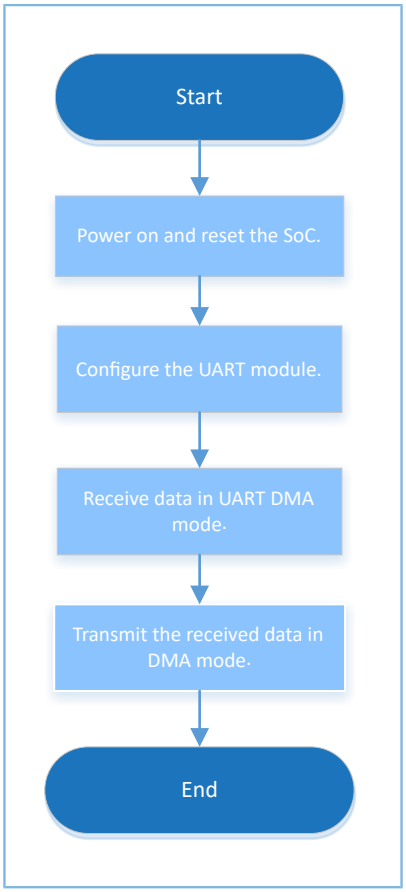

Figure 3-32 UART DMA example workflow

1. Configure the UART module.

```
g_uart_handle.p_instance         = SERIAL_PORT_GRP;
g_uart_handle.init.baud_rate     = 115200;
g_uart_handle.init.data_bits     = UART_DATABITS_8;
g_uart_handle.init.stop_bits     = UART_STOPBITS_1;
g_uart_handle.init.parity        = UART_PARITY_NONE;
g_uart_handle.init.hw_flow_ctrl = UART_HWCONTROL_NONE;
g_uart_handle.init.rx_timeout_mode = UART_RECEIVER_TIMEOUT_ENABLE;
hal_uart_deinit(&g_uart_handle);
hal_uart_init(&g_uart_handle);
```

- init.baud_rate: baud rate of the UART (bps)
- init.data_bits: data bit of the UART
- init.stop_bits: stop bit of the UART
- init.parity: odd-even parity bit of UART
- init.hw_flow_ctrl: enablement bit for hardware flow control
- init.rx_timeout_mode: reception timeout enablement bit

2. Call hal_uart_transmit_dma() to transmit data. Reception results and status are returned by calling hal_uart_rx_cplt_callback(). Users can customize executable operations in this callback function. Due to the

non-blocking APIs adopted, judge whether the transmission/reception is completed by using the while loop in subsequent operations. The code snippet is as follows:

```
void hal_uart_rx_cplt_callback(uart_handle_t *p_uart)
{
    rdone = 1;
    rlen = p_uart->rx_xfer_size - p_uart->rx_xfer_count;
    memcpy(g_tdata, g_rdata, rlen);
    memset(g_rdata, 0, sizeof(g_rdata));
}

void hal_uart_tx_cplt_callback(uart_handle_t *p_uart)
{
    tdone = 1;
}

void hal_uart_error_callback(uart_handle_t *p_uart)
{
    tdone = 1;
    rdone = 1;
}
hal_uart_transmit_dma(&g_uart_handle, g_print_str1, 55);
while (hal_uart_get_state(&g_uart_handle) ! = HAL_UART_STATE_READY);
hal_uart_transmit_dma(&g_uart_handle, g_print_str2, 8);
while (hal_uart_get_state(&g_uart_handle) ! = HAL_UART_STATE_READY);
```

3. Call hal_uart_transmit_dma() to transmit the received data. Transmission results and status are returned by calling hal_uart_tx_cplt_callback(). When the received data is “0”, the test ends. The code snippet is as follows:

```
do {
    rdone = 0;
    hal_uart_receive_dma(&g_uart_handle, g_rdata, sizeof(g_rdata));
    while (0 == rdone);
    tdone = 0;
    hal_uart_transmit_dma(&g_uart_handle, g_tdata, rlen);
    while (0 == tdone);
} while (g_tdata[0] ! = '0');
```

3.13.1.2 Test and Verification

1. Download *uart_dma_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the data transmitted and received through the UART in the **Send Data** pane and the **Receive Data** pane respectively on GRUart.

3.13.2 UART Interrupt

The UART Interrupt example project enables data RX and TX through UART in interrupt mode.

The source code and project file of the UART Interrupt example are in SDK_Folder\projects\peripheral\uart\uart_interrupt, and project file is in the Keil_5 folder.

3.13.2.1 Code Interpretation

Figure 3-33 shows the workflow of a UART INTERRUPT example project.

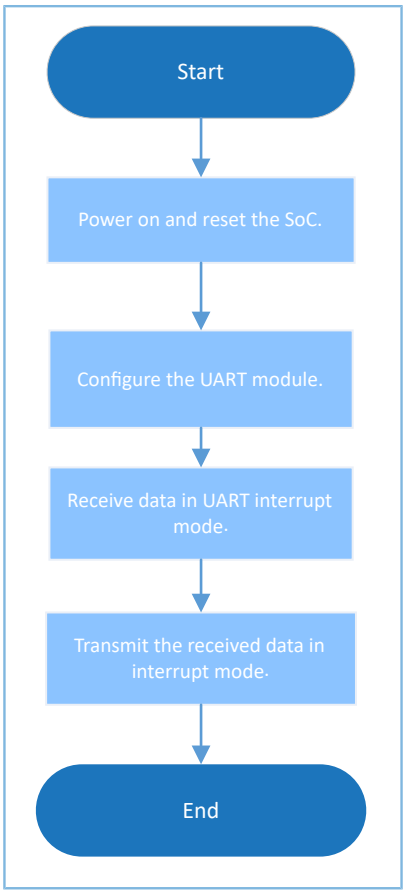

Figure 3-33 UART Interrupt example workflow

1. Configure the UART module.

```
g_uart_handle.p_instance = SERIAL_PORT_GRP;
g_uart_handle.init.baud_rate = 115200;
g_uart_handle.init.data_bits = UART_DATABITS_8;
g_uart_handle.init.stop_bits  = UART_STOPBITS_1;
g_uart_handle.init.parity = UART_PARITY_NONE;
g_uart_handle.init.hw_flow_ctrl = UART_HWCONTROL_NONE;
g_uart_handle.init.rx_timeout_mode = UART_RECEIVER_TIMEOUT_ENABLE;
hal_uart_deinit(&g_uart_handle);
hal_uart_init(&g_uart_handle);
```

   For more information about UART configurations, see “Section 3.13.1 UART DMA”.

2. Call hal_uart_transmit_it() to transmit data. Reception results and status are returned by calling hal_uart_rx_cplt_callback(). Users can customize executable operations in this callback function. Due to the non-blocking APIs adopted, judge whether the transmission/reception is completed by using the while loop in subsequent operations.

```
hal_uart_transmit_it(&g_uart_handle, (uint8_t *)"\r\nPlease input characters(<126) and end
 with newline.\r\n", 55);
```



```
while (!g_tx_done);
g_tx_done = 0;
hal_uart_transmit_it(&g_uart_handle, (uint8_t *)"Input:\r\n", 8);
while (!g_tx_done);
```

3. Call hal_uart_receive_it() to receive data. Transmission results and status are returned by calling hal_uart_rx_cplt_callback(). Call hal_uart_receive_it() to continue receiving data in the reception complete callback, and transmit the received data by calling hal_uart_transmit_it(). Transmission results and status are returned by calling hal_uart_tx_cplt_callback(). When the received data is “0”, the test ends. The code snippet is as follows:

```
void hal_uart_tx_cplt_callback(uart_handle_t *huart)
{
    g_tx_done = 1;
}

void hal_uart_rx_cplt_callback(uart_handle_t *huart)
{
    g_rx_done = 1;
    g_rx_len  = g_uart_handle.rx_xfer_size - g_uart_handle.rx_xfer_count;
    hal_uart_receive_it(&g_uart_handle, rx_data, 128);
}

do {
    while (!g_rx_done);
    g_rx_done = 0;
    g_tx_done = 0;
    hal_uart_transmit_it(&g_uart_handle, rx_data, g_rx_len);
    while (!g_tx_done);
} while (rx_data[0] ! = '0');
```

3.13.2.2 Test and Verification

1. Download *uart_interrupt_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the data transmitted and received through the UART in the **Receive Data** pane on GRUart.

3.13.3 UART TX & RX

The UART TX & RX example project enables data RX and TX through UART in polling mode.

The source code and project file of the UART TX & RX example are in SDK_Folder\projects\peripheral\uart\uart_tx_rx, and project file is in the Keil_5 folder.

3.13.3.1 Code Interpretation

Figure 3-34 shows the workflow of a UART TX & RX example project.

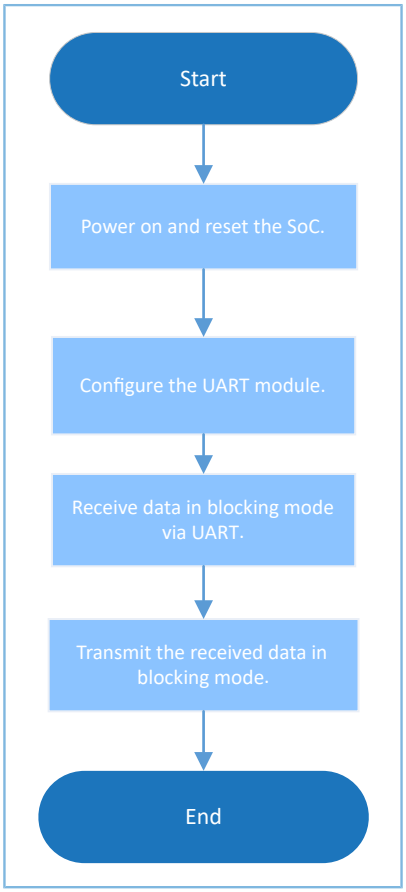

Figure 3-34 UART TX and RX example workflow

1. Configure the UART module.

```
g_uart_handle.p_instance  = SERIAL_PORT_GRP;
g_uart_handle.init.baud_rate = 115200;
g_uart_handle.init.data_bits  = UART_DATABITS_8;
g_uart_handle.init.stop_bits  = UART_STOPBITS_1;
g_uart_handle.init.parity  = UART_PARITY_NONE;
g_uart_handle.init.hw_flow_ctrl = UART_HWCONTROL_NONE;
g_uart_handle.init.rx_timeout_mode = UART_RECEIVER_TIMEOUT_ENABLE;
hal_uart_deinit(&g_uart_handle);
hal_uart_init(&g_uart_handle);
```

   For more information about UART configurations, see “Section 3.13.1 UART DMA”.

2. Call hal_uart_receive() to receive data in polling mode. After the RX is completed, transmit the received data by calling hal_uart_transmit(). When the received data is “0”, the test ends. The code snippet is as follows:

```
do {
    hal_uart_receive(&g_uart_handle, rdata, 128, 20);
    rlen = g_uart_handle.rx_xfer_size - g_uart_handle.rx_xfer_count;
    hal_uart_transmit(&g_uart_handle, rdata, rlen, 1000);
} while (rdata[0] ! = '0');
```

3.13.3.2 Test and Verification

1. Download u*art_tx_rx_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the data transmitted and received through the UART in the **Receive Data** pane on GRUart.

3.14 DMA

Direct Memory Access (DMA) is a mechanism that allows quick data transfer, and enables direct data read/write between peripherals and memories. The central processing unit (CPU) only takes part when data transfer starts and ends (in handling interrupts), and is available for other operations during data transfer.

3.14.1 DMA Memory to Memory

The DMA Memory to Memory example project enables data transfer from one memory to another in DMA mode.

The source code and project file of the DMA Memory to Memory example are in SDK_`Folder\projects\peripheral\dma\dma_memory_to_memory`, and project file is in the Keil_5 folder.

3.14.1.1 Code Interpretation

Figure 3-35 shows the workflow of a DMA Memory to Memory example project.

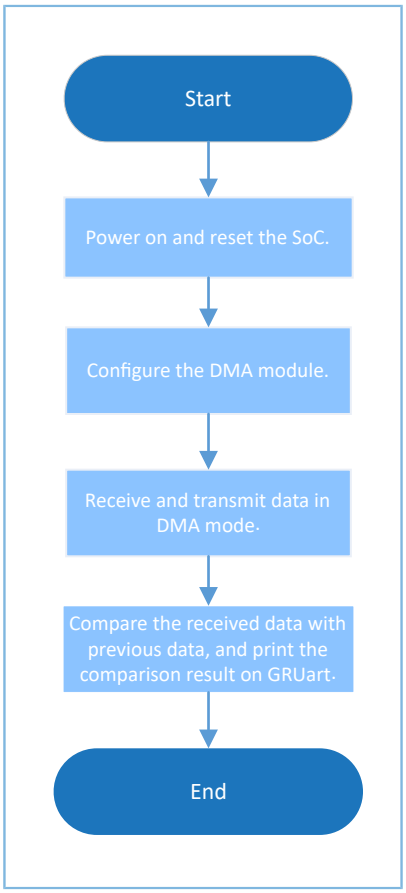

Figure 3-35 DMA Memory to Memory example workflow

1. Configure the DMA module.

```
g_dma_handle.channel = DMA_Channel0;
g_dma_handle.init.direction = DMA_MEMORY_TO_MEMORY;
g_dma_handle.init.src_increment = DMA_SRC_INCREMENT;
g_dma_handle.init.dst_increment = DMA_DST_INCREMENT;
g_dma_handle.init.src_data_alignment = DMA_SDATAALIGN_BYTE;
g_dma_handle.init.dst_data_alignment = DMA_DDATAALIGN_BYTE;
g_dma_handle.init.mode = DMA_NORMAL;
g_dma_handle.init.priority = DMA_PRIORITY_LOW;

hal_dma_init(&g_dma_handle);
```

   - channel: DMA channel. Options: DMA_Channel0 to DMA_Channel7
   - direction: data transfer direction. Options: DMA_MEMORY_TO_MEMORY, DMA_MEMORY_TO_PERIPH, DMA_PERIPH_TO_MEMORY, and DMA_PERIPH_TO_PERIPH
   - src_increment: method of updating registers of the source address. Options: DMA_SRC_INCREMENT, DMA_SRC_DECREMENT, and DMA_SRC_NO_CHANGE
   - dst_increment: method of updating registers of the destination address. Options: DMA_DST_INCREMENT, DMA_DST_DECREMENT, and DMA_DST_NO_CHANGE
   - src_data_alignment: source data width. Options: DMA_SDATAALIGN_BYTE, DMA_SDATAALIGN_HALFWORD, and DMA_SDATAALIGN_WORD
   - dst_data_alignment: destination data width. Options: DMA_DDATAALIGN_BYTE, DMA_DDATAALIGN_HALFWORD, and DMA_DDATAALIGN_WORD
   - mode: DMA operation mode. Options: DMA_NORMAL and DMA_CIRCULAR
   - priority: software priority of DMA channel. Options: DMA_PRIORITY_LOW, DMA_PRIORITY_MEDIUM, and DMA_PRIORITY_HIGH

2. Call the blocking API hal_dma_start() to start data transfer; call hal_dma_poll_for_transfer() to complete data reception/transmission. The code snippet is as follows:

```
hal_dma_start(&g_dma_handle, (uint32_t)&g_src_data, (uint32_t)&g_dst_data, DMA_DATA_LEN);
hal_dma_poll_for_transfer(&g_dma_handle, 10);
```

3. Call hal_dma_start_it() to start data transfer in interrupt mode. Transfer results are returned by calling dma_tfr_callback(), dma_blk_callback(), dma_err_callback(), or dma_abort_callback(). Users can customize executable operations in the functions above. The code snippet is as follows:

```
void dma_tfr_callback(struct _dma_handle * hdma)
{
    g_tfr_flag = 1;
    //    printf("tfr interrupt\r\n");
}

void dma_blk_callback(struct _dma_handle * hdma)
{
    g_blk_flag = 1;
    //    printf("blk interrupt\r\n");
}
```

```
void dma_err_callback(struct _dma_handle * hdma)
{
    g_err_flag = 1;
    //    printf("err interrupt\r\n");
}

void dma_abort_callback(struct _dma_handle * hdma)
{
    g_abt_flag = 1;
    //    printf("abort interrupt\r\n");
}

hal_dma_start(&g_dma_handle, (uint32_t)&g_src_data, (uint32_t)&g_dst_data, DMA_DATA_LEN);
```

3.14.1.2 Test and Verification

1. Download *dma_m2m_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the comparison between the original data and the data transferred via DMA in the **Receive Data** pane on GRUart.

3.15 QSPI

Queued SPI (QSPI) is a type of SPI that introduces a queue transfer mechanism. It is a specialized communications interface that connects to a flash memory with one, two, or four cables in SPI mode. A QSPI is commonly used to connect to an external flash memory.

3.15.1 QSPI Flash

The QSPI Flash example project enables a QSPI to connect to an external flash memory, and read/write data.

The source code and project file of the QSPI Flash example are in `SDK_Folder\projects\peripheral\qspi\qspi_flash`, and project file is in the Keil_5 folder.

3.15.1.1 Code Interpretation

Figure 3-36 shows the workflow of a QSPI Flash example project.

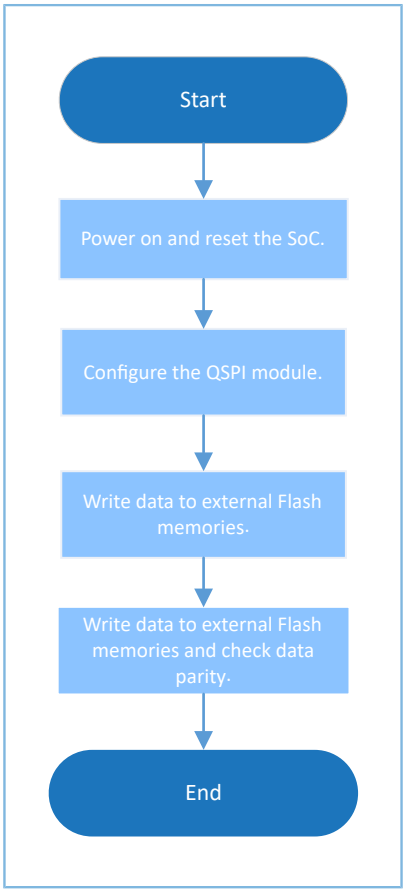

Figure 3-36 QSPI Flash example workflow

1. Configure the QSPI module.

```
g_qspi_handle.init.clock_prescaler = SystemCoreClock / 1000000;
g_qspi_handle.init.clock_mode = QSPI_CLOCK_MODE_0;
g_qspi_handle.init.rx_sample_delay = 0;

hal_qspi_deinit(&g_qspi_handle);
if (HAL_OK != hal_qspi_init(&g_qspi_handle))
{
    printf("\r\nFlash initial failed.\r\n");
    return 0;
}
```

- init.clock_prescaler: This parameter sets clock frequency of the QSPI.
- init.clock_mode: This parameter sets clock mode of the QSPI. Options: QSPI_CLOCK_MODE_0, QSPI_CLOCK_MODE_1, QSPI_CLOCK_MODE_2, and QSPI_CLOCK_MODE_3

2. Initialize the QSPI module by calling hal_qspi_init(). The code snippet is as follows:

```
hal_qspi_init(&g_qspi_handle)
```

3. Get the ID of a flash memory by calling SPI_FLASH_Read_Device_ID(). The code snippet is as follows:

```
device_id = SPI_FLASH_Read_Device_ID();
```

```
printf("Read_Device_ID = 0x%06X\r\n", device_id);
```

4. Write data to a flash memory by calling SPI_FLASH_Page_Program(). The code snippet is as follows:

```
printf("Page_Program...\r\n");
SPI_FLASH_Page_Program(FLASH_PROGRAM_START_ADDR, write_buffer);
printf("Page Program Success.\r\n");
```

5. Read data from a flash memory by calling SPI_FLASH_Read(). The code snippet is as follows:

```
SPI_FLASH_Read(FLASH_PROGRAM_START_ADDR, read_buffer, FLASH_OPERATION_LENGTH);
```

6. Enable four-cable-connection mode by calling SPI_FLASH_Enable_Quad(). The code snippet is as follows:

```
SPI_FLASH_Enable_Quad();
```

7. Call the APIs below to enable four-cable-connection mode of a QSPI. Users can view details by checking the source code in *spi_flash.c.*

```
SPI_FLASH_Dual_Output_Fast_Read(FLASH_PROGRAM_START_ADDR, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("Dual_Output_Fast_Read." );
qspi_flash_memcmp(FLASH_PROGRAM_START_ADDR, write_buffer, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("\r\n");

SPI_FLASH_Dual_IO_Fast_Read(FLASH_PROGRAM_START_ADDR, read_buffer, FLASH_OPERATION_LENGTH);
printf("Dual_IO_Fast_Read." );
qspi_flash_memcmp(FLASH_PROGRAM_START_ADDR, write_buffer, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("\r\n");

SPI_FLASH_Quad_Output_Fast_Read(FLASH_PROGRAM_START_ADDR, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("Quad_Output_Fast_Read." );
qspi_flash_memcmp(FLASH_PROGRAM_START_ADDR, write_buffer, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("\r\n");

SPI_FLASH_Quad_IO_Fast_Read(FLASH_PROGRAM_START_ADDR, read_buffer, FLASH_OPERATION_LENGTH);
printf("Quad_IO_Fast_Read." );
qspi_flash_memcmp(FLASH_PROGRAM_START_ADDR, write_buffer, read_buffer,
 FLASH_OPERATION_LENGTH);
printf("\r\n");
```

3.15.1.2 Test and Verification

1. Download *qspi_flash_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view data about QSPI operations on the flash memory in the **Receive Data** pane on GRUart.

3.15.2 QSPI DMA SPI

The QSPI DMA SPI example enables connection to an external flash memory and data read from the flash memory through QSPI, and data transfer through SPI Master DMA mode.

The source code and project file of the QSPI DMA SPI example are in `SDK_Folder\projects\peripheral\qspi\qspi_flash_dma_spim`, and project file is in the Keil_5 folder.

3.15.2.1 Code Interpretation

Figure 3-37 shows the workflow of a QSPI DMA SPI example project.

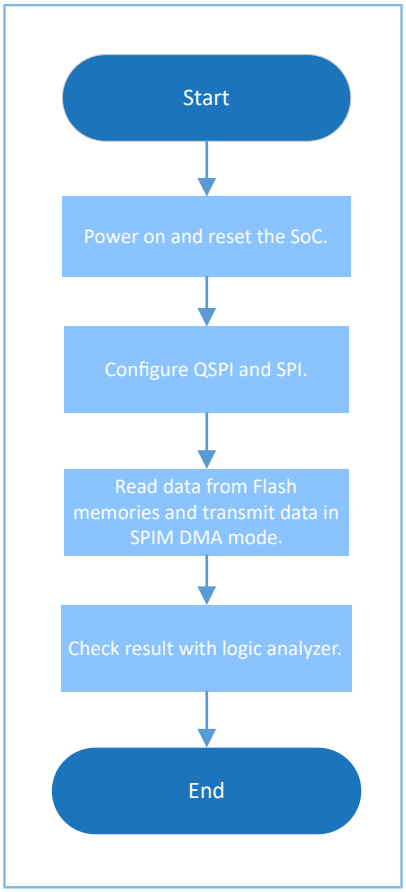

Figure 3-37 QSPI DMA SPI example workflow

1. Configure the QSPI DMA SPI module.

```
g_qspi_handle.init.clock_prescaler = SystemCoreClock / 1000000;
g_qspi_handle.init.clock_mode = QSPI_CLOCK_MODE_3;
g_qspi_handle.init.rx_sample_delay = 0;

hal_qspi_deinit(&g_qspi_handle);
if (HAL_OK != hal_qspi_init(&g_qspi_handle))
{
    printf("\r\nFlash initial failed.\r\n");
    return 0;
}

p_spi_handle->init.data_size          = SPI_DATASIZE_32BIT;
```



```
p_spi_handle->init.clock_polarity     = SPI_POLARITY_LOW;
p_spi_handle->init.clock_phase        = SPI_PHASE_1EDGE;
p_spi_handle->init.baudrate_prescaler = SystemCoreClock / 1000000;
p_spi_handle->init.ti_mode            = SPI_TIMODE_DISABLE;
p_spi_handle->init.slave_select       = SPI_SLAVE_SELECT_0;
hal_spi_deinit(p_spi_handle);
hal_spi_init(p_spi_handle);
```

(1). Configure the QSPI module.

Configure the clock frequency of the QSPI to 1 MHz, and the clock signal to a high level when the QSPI is idle.

(2). Configure the SPI module.

- init.data_size: size of data transmitted through SPI. Options: SPI_DATASIZE_4BIT to SPI_DATASIZE_32BIT
- init.clock_polarity: clock signal when the SPI is idle. Options: SPI_POLARITY_LOW and SPI_POLARITY_HIGH
- init.clock_phase: SPI clock switch time. Options: SPI_PHASE_1EDGE and SPI_PHASE_2EDGE
- baudrate_prescaler: SPI clock. The value is SystemCoreClock/1000000 (1 MHz).
- init.ti_mode: enabling the TI mode of the SPI or not. Options: SPI_TIMODE_DISABLE and SPI_TIMODE_ENABLE
- init.slave_select: This parameter sets an SPI slave. Options: SPI_SLAVE_SELECT_0, SPI_SLAVE_SELECT_1, and SPI_SLAVE_SELECT_ALL

2. Initialize the QSPI module by calling hal_qspi_init(). The code snippet is as follows:

```
hal_qspi_init(&g_qspi_handle);
```

3. Get the ID of a flash memory by calling SPI_FLASH_Read_Device_ID(). The code snippet is as follows:

```
device_id = SPI_FLASH_Read_Device_ID();
printf("Read_Device_ID = 0x%06X\r\n", device_id);
```

4. Write data to a flash memory by calling SPI_FLASH_Page_Program(). The code snippet is as follows:

```
printf("Page_Program...\r\n");
SPI_FLASH_Page_Program(FLASH_PROGRAM_START_ADDR, flash_buffer);
printf("Page Program Success.\r\n");
```

5. Set the QSPI in SPI mode, and the data to 32 bits wide. The transfer destination is EEPROM, and the output data size is (FLASH_PAGE_SIZE >> 2) – 1. The code snippet is as follows:

```
/* Config QSPI in SPI mode, data size is 32bits */
__HAL_QSPI_DISABLE(p_qspi_handle);
ll_spi_set_frame_format(p_qspi_handle->p_instance, LL_SSI_FRF_SPI);
ll_spi_set_data_size(p_qspi_handle->p_instance, LL_SSI_DATASIZE_32BIT);
ll_spi_set_transfer_direction(p_qspi_handle->p_instance, LL_SSI_READ_EEPROM);
ll_spi_set_receive_size(p_qspi_handle->p_instance, (FLASH_PAGE_SIZE >> 2) - 1);
```

```
__HAL_QSPI_ENABLE(p_qspi_handle);
```

6. Set the SPI mode as write. The code snippet is as follows:

```
/* Config SPI in write mode */
__HAL_SPI_DISABLE(p_spi_handle);
ll_spi_set_transfer_direction(p_spi_handle->p_instance, LL_SSI_SIMPLEX_TX);
__HAL_SPI_ENABLE(p_spi_handle);
```

7. Configure the DMA burst length. The code snippet is as follows:

```
/* Config burst length of DMA */
ll_dma_set_source_burst_length(DMA, p_qspi_handle->p_dma->channel,
 LL_DMA_SRC_BURST_LENGTH_1);
ll_dma_set_destination_burst_length(DMA, p_qspi_handle->p_dma->channel,
 LL_DMA_DST_BURST_LENGTH_4);
```

8. Start DMA transfer and wait until the transfer completes. The code snippet is as follows:

```
/* Start read SPI flash */
uint32_t cmd_addr = ((uint32_t)SPI_FLASH_CMD_READ << 24) | FLASH_PROGRAM_START_ADDR;
hal_dma_start(p_qspi_handle->p_dma, (uint32_t)& p_qspi_handle->p_instance->DATA,
 (uint32_t)&p_spi_handle->p_instance->DATA, FLASH_PAGE_SIZE >> 2);
p_qspi_handle->p_instance->DATA = cmd_addr;
__HAL_QSPI_ENABLE_DMARX(p_qspi_handle);
__HAL_SPI_ENABLE_DMATX(p_spi_handle);
hal_status_t status = hal_dma_poll_for_transfer(p_qspi_handle->p_dma, 1000);
if (status == HAL_OK)
{
    printf("Transfer finished.\r\n");
}
else
{
    printf("Transfer failed, hal_status = %d\r\n", status);
}
```

3.15.2.2 Test and Verification

1. Download *qspi_flash_dma_spim_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view data interactions of QSPI FLASH SPI in the **Receive Data** pane on GRUart.

3.15.3 QSPI DMA UART

The QSPI DMA UART example enables connection to an external flash memory through a QSPI interface, data read from the flash memory, and data transfer in UART DMA mode.

The source code and project file of the QSPI DMA UART example are in `SDK_Folder\projects\peripheral\qspi\qspi_flash_dma_uart`, and project file is in the Keil_5 folder.

3.15.3.1 Code Interpretation

Figure 3-38 shows the workflow of a QSPI DMA UART example project.

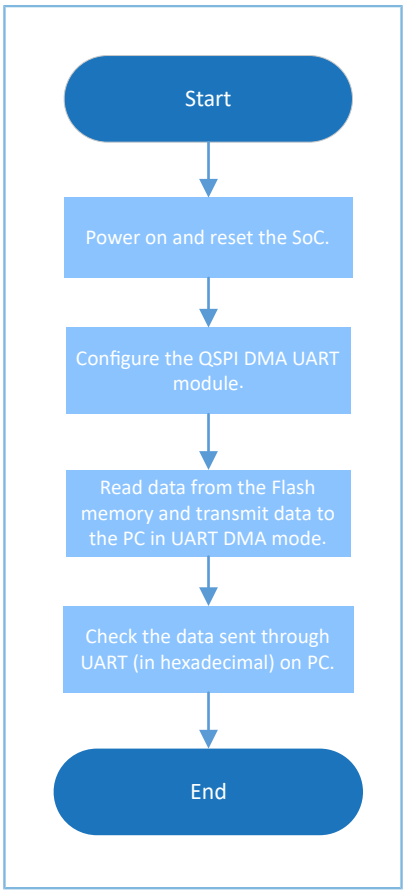

Figure 3-38 QSPI DMA UART example workflow

1. Configure the QSPI DMA UART module.

```
g_qspi_handle.init.clock_prescaler = SystemCoreClock / 1000000;
g_qspi_handle.init.clock_mode = QSPI_CLOCK_MODE_3;
g_qspi_handle.init.rx_sample_delay = 0;

hal_qspi_deinit(&g_qspi_handle);
if (HAL_OK != hal_qspi_init(&g_qspi_handle))
{
    printf("\r\nFlash initial failed.\r\n");
    return 0;
}

uart_handle.p_instance         = UART1;
uart_handle.init.baud_rate     = 115200;
uart_handle.init.data_bits     = UART_DATABITS_8;
uart_handle.init.stop_bits     = UART_STOPBITS_1;
uart_handle.init.parity        = UART_PARITY_NONE;
uart_handle.init.hw_flow_ctrl  = UART_HWCONTROL_NONE;
uart_handle.init.rx_timeout_mode = UART_RECEIVER_TIMEOUT_DISABLE;
hal_uart_deinit(&uart_handle);
hal_uart_init(&uart_handle);
```

(1). Configure the QSPI module.

Configure the clock frequency of the QSPI to 1 MHz, and the clock signal to a high level when the QSPI is idle.

(2). Configure the UART module.

For more information about UART configurations, see "Section 3.13.1 UART DMA". Set UART to UART1.

2. Initialize the QSPI module by calling hal_qspi_init(). The code snippet is as follows:

```
hal_qspi_init(&g_qspi_handle)
```

3. Get the ID of a flash memory by calling SPI_FLASH_Read_Device_ID(). The code snippet is as follows:

```
device_id = SPI_FLASH_Read_Device_ID();
printf("Read_Device_ID = 0x%06X\r\n", device_id);
```

4. Read data from a flash memory by calling SPI_FLASH_Read(). The code snippet is as follows:

```
SPI_FLASH_Read(FLASH_PROGRAM_START_ADDR, flash_buffer, FLASH_PAGE_SIZE);
for (i = 0; i < FLASH_PAGE_SIZE; i++) {
    if (flash_buffer[i] ! = 0xFF)
        break;
}

if (i < FLASH_PAGE_SIZE)
{
    printf("Erase Sector...\r\n");
    SPI_FLASH_Sector_Erase(FLASH_PROGRAM_START_ADDR);
    printf("Erase Sector Success.\r\n");
}
```

5. Write data to a flash memory by calling SPI_FLASH_Page_Program(). The code snippet is as follows:

```
printf("Page_Program...\r\n");
SPI_FLASH_Page_Program(FLASH_PROGRAM_START_ADDR, flash_buffer);
printf("Page Program Success.\r\n");
```

6. Set the QSPI to QSPI mode, and the data to 32 bits wide. The transfer destination is EEPROM, and the output data size is (FLASH_PAGE_SIZE >> 2) – 1. The code snippet is as follows:

```
/* Config QSPI in SPI mode, data size is 32bits */
__HAL_QSPI_DISABLE(p_qspi_handle);
ll_spi_set_frame_format(p_qspi_handle->p_instance, LL_SSI_FRF_SPI);
ll_spi_set_data_size(p_qspi_handle->p_instance, LL_SSI_DATASIZE_8BIT);
ll_spi_set_transfer_direction(p_qspi_handle->p_instance, LL_SSI_READ_EEPROM);
ll_spi_set_receive_size(p_qspi_handle->p_instance, FLASH_PAGE_SIZE - 1);
__HAL_QSPI_ENABLE(p_qspi_handle);
```

7. Configure the FIFO threshold and DMA burst length of UART1. The code snippet is as follows:

```
/* Config FIFO threshold of UART and burst length of DMA */
ll_uart_set_tx_fifo_threshold(uart_handle.p_instance, LL_UART_TX_FIFO_TH_QUARTER_FULL);
ll_uart_enable_it(uart_handle.p_instance, UART_IT_THRE);
```

```
ll_dma_set_source_burst_length(DMA, p_qspi_handle->p_dma->channel,
 LL_DMA_SRC_BURST_LENGTH_1);
ll_dma_set_destination_burst_length(DMA, p_qspi_handle->p_dma->channel,
 LL_DMA_DST_BURST_LENGTH_8);
```

8. Start DMA transfer and wait until the transfer completes. The code snippet is as follows:

```
uint32_t cmd_addr = ((uint32_t)SPI_FLASH_CMD_READ << 24) | FLASH_PROGRAM_START_ADDR;
hal_dma_start(p_qspi_handle->p_dma, (uint32_t)& p_qspi_handle->p_instance->DATA,
 (uint32_t)&UART1->RBR_DLL_THR, FLASH_PAGE_SIZE);
p_qspi_handle->p_instance->DATA = (cmd_addr >> 24 ) & 0xff;
p_qspi_handle->p_instance->DATA = (cmd_addr >> 16 ) & 0xff;
p_qspi_handle->p_instance->DATA = (cmd_addr >>  8 ) & 0xff;
p_qspi_handle->p_instance->DATA = (cmd_addr >>  0 ) & 0xff;

__HAL_QSPI_ENABLE_DMARX(p_qspi_handle);
ll_spi_enable_ss(p_qspi_handle->p_instance, LL_SSI_SLAVE0);
hal_status_t status = hal_dma_poll_for_transfer(p_qspi_handle->p_dma, 1000);
if (status == HAL_OK)
{
   printf("Transfer finished.\r\n");
}
else
{
    printf("Transfer failed, hal_status = %d\r\n", status);
}
```

3.15.3.2 Test and Verification

1. Download *qspi_flash_dma_uart_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view data interactions of QSPI FLASH UART in the **Receive Data** pane on GRUart.

3.16 Timer

A timer can trigger a specific event or task during a specific time interval for more than once. It is in general use in timing tasks.

3.16.1 Dual Timer

The Dual Timer example project enables regular printing of program information with Dual Timer in interrupt mode.

The source code and project file of the Dual Timer example are in SDK_Folder\projects\peripheral\timer\dual_timer, and project file is in the Keil_5 folder.

3.16.1.1 Code Interpretation

Figure 3-39 shows the workflow of a Dual Timer example project.

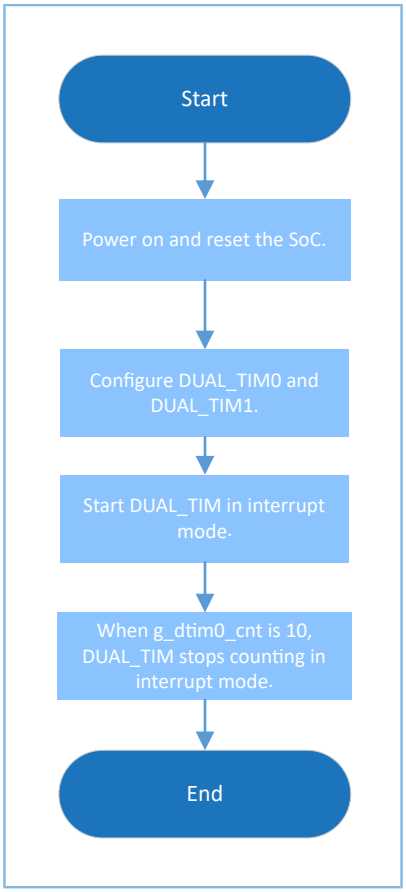

Figure 3-39 Dual Timer example workflow

1. Configure the Dual Timer module.

```
g_dual_tim0_handle.p_instance = DUAL_TIMER0;
g_dual_tim0_handle.init.prescaler   = DUAL_TIM_PRESCALER_DIV0;
g_dual_tim0_handle.init.counter_mode = DUAL_TIM_COUNTERMODE_LOOP;
g_dual_tim0_handle.init.auto_reload  = SystemCoreClock - 1;
hal_dual_timer_base_init(&g_dual_tim0_handle);

g_dual_tim1_handle.p_instance = DUAL_TIMER1;
g_dual_tim1_handle.init.prescaler   = DUAL_TIM_PRESCALER_DIV0;
g_dual_tim1_handle.init.counter_mode = DUAL_TIM_COUNTERMODE_LOOP;
g_dual_tim1_handle.init.auto_reload = SystemCoreClock / 100 - 1;
```

   - prescaler: prescaler. Options: DUAL_TIM_PRESCALER_DIV0, DUAL_TIM_PRESCALER_DIV16, and DUAL_TIM_PRESCALER_DIV256
   - counter_mode: counter mode. Options: DUAL_TIM_COUNTERMODE_LOOP and DUAL_TIM_COUNTERMODE_ONESHOT
   - auto_reload: automatic reload value. DUAL_TIMER0 indicates to load every second, and DUAL_TIMER1 indicates to load every 10 milliseconds.

2. Call hal_dual_timer_base_start_it() to start counting in interrupt mode. auto_reload receives timing, which is returned by calling hal_dual_timer_period_elapsed_callback(). The code snippet is as follows:

```
void hal_dual_timer_period_elapsed_callback(dual_timer_handle_t *hdtim)
{
    if (hdtim->p_instance == DUAL_TIMER0)
    {
        printf("\r\nThis is %dth call DUALTIM0.\r\n", g_dtim0_cnt++);
    }
    else
    {
        g_dtim1_cnt++;
    }
}

hal_dual_timer_base_start_it(&g_dual_tim1_handle);
hal_dual_timer_base_start_it(&g_dual_tim0_handle);
```

3. Call hal_dual_timer_base_stop_it() to stop counting in interrupt mode. The code snippet is as follows:

```
hal_dual_timer_base_stop_it(&g_dual_tim1_handle);
hal_dual_timer_base_stop_it(&g_dual_tim0_handle);
```

3.16.1.2 Test and Verification

1. Download *dual_timer_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the debugging information printed every second in the **Receive Data** pane on GRUart. The program stops running after printing for 10 times.

3.16.2 Timer

The Timer example project enables regular printing of program information with Timer in interrupt mode.

The source code and project file of the Timer example are in SDK_Folder\projects\peripheral\timer\timer, and project file is in the Keil_5 folder.

3.16.2.1 Code Interpretation

Figure 3-40 shows the workflow of a Timer example project.

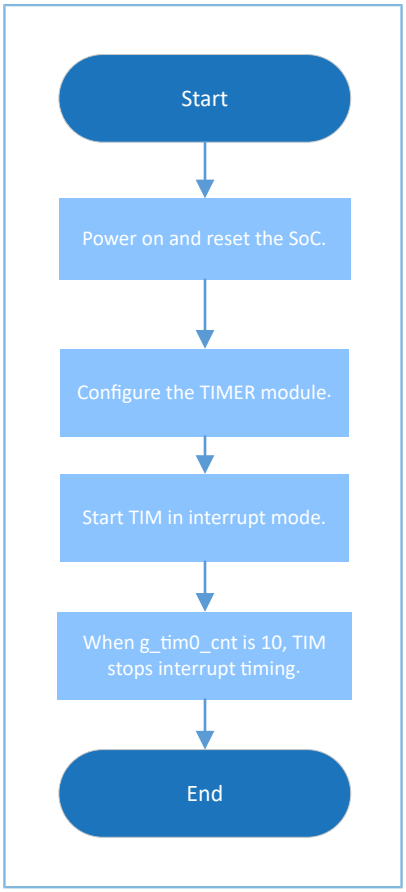

Figure 3-40 Timer example workflow

1. Configure the Timer module.

```
g_tim0_handle.p_instance = TIMER0;
g_tim0_handle.init.auto_reload = SystemCoreClock - 1;
hal_timer_base_init(&g_tim0_handle);

g_tim1_handle.p_instance = TIMER1;
g_tim1_handle.init.auto_reload = SystemCoreClock / 100 - 1;
hal_timer_base_init(&g_tim1_handle);
```

- auto_reload: automatic reload value. TIMER0 indicates to load every second, and TIMER1 indicates to load every 10 milliseconds.

2. Call hal_timer_base_start_it() to start counting in interrupt mode. When the time is up (set by auto_reload), the result is returned by calling hal_timer_period_elapsed_callback(). The code snippet is as follows:

```
void hal_timer_period_elapsed_callback(timer_handle_t *htim)
{
    if (htim->p_instance == TIMER0)
    {
        printf("\r\nThis is %dth call TIMER0.\r\n", g_tim0_cnt++);
    }
    else
    {
        g_tim1_cnt++;
    }
```

```
}

hal_timer_base_start_it(&g_tim1_handle);
hal_timer_base_start_it(&g_tim0_handle);
```

3. Call hal_timer_base_stop_it() to stop counting in interrupt mode. The code snippet is as follows:

```
hal_timer_base_stop_it(&g_tim1_handle);
hal_timer_base_stop_it(&g_tim0_handle);
```

3.16.2.2 Test and Verification

1. Download *timer_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the debugging information printed every second in the **Receive Data** pane on GRUart. The program stops running after printing for 10 times.

3.16.3 Timer Wakeup

The Timer Wakeup example project enables to wake up the SoC with Dual Timer in interrupt mode.

The source code and project file of the Timer Wakeup example are in SDK_Folder\projects\peripheral\timer\timer_wakeup, and project file is in the Keil_5 folder.

3.16.3.1 Code Interpretation

Figure 3-41 shows the workflow of a Timer Wakeup example project.

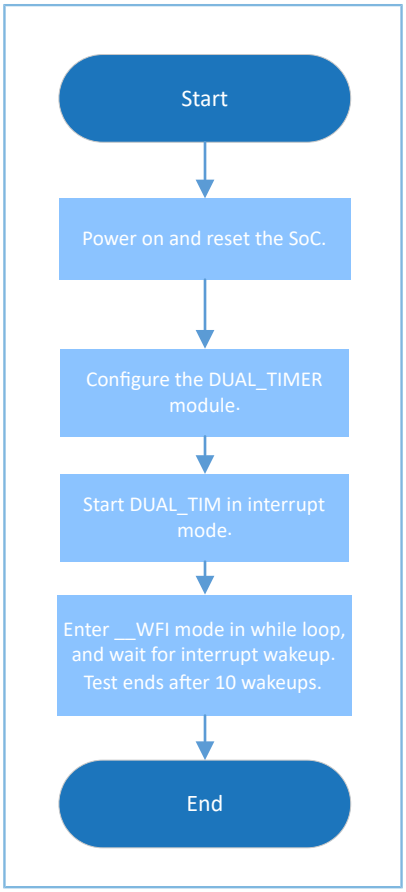

Figure 3-41 Timer Wakeup example workflow

1. Configure the Dual Timer module.

```
g_dual_tim0_handle.p_instance = DUAL_TIMER0;
g_dual_tim0_handle.init.prescaler   = DUAL_TIM_PRESCALER_DIV0;
g_dual_tim0_handle.init.counter_mode = DUAL_TIM_COUNTERMODE_LOOP;
g_dual_tim0_handle.init.auto_reload  = SystemCoreClock - 1;
hal_dual_timer_base_init(&g_dual_tim0_handle);
```

   For more information about Dual Timer configurations, see “Section 3.16.1 Dual Timer”.

2. Call hal_dual_timer_base_start_it() to enable the timer to wake up the SoC in interrupt mode. Call auto_reload to set the time to wake up the SoC, and the result is returned by calling hal_dual_timer_period_elapsed_callback(). The code snippet is as follows:

```
void hal_dual_timer_period_elapsed_callback(dual_timer_handle_t *hdtim)
{
    if (hdtim->p_instance == DUAL_TIMER0)
    {
        g_dtim0_cnt++;
    }
}

hal_dual_timer_base_start_it(&g_dual_tim0_handle);
```

3. The SoC enters sleep mode. The code snippet is as follows:

```
while (g_dtim0_cnt < 10)
{
    printf("\r\nEnter sleep at timer counter = %d\r\n", g_dtim0_cnt);
    SCB->SCR |= 0x04;
    __WFI();
    printf("Wakeup from sleep at timer counter = %d\r\n", g_dtim0_cnt);
}
```

4. Call hal_dual_timer_base_stop_it() to stop counting in interrupt mode. The code snippet is as follows:

```
hal_dual_timer_base_stop_it(&g_dual_tim0_handle);
```

3.16.3.2 Test and Verification

1. Download *timer_wakeup_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the debugging information printed every second in the **Receive Data** pane on GRUart. The program stops running after printing for 10 times.

3.17 WDT

A watchdog timer (WDT) is a hardware timer that helps an SoC recover from malfunctions (such as a hang or failure to feed the watchdog in time) and operate normally by sending reset/restart/shut-down signals. A WDT is generally used to reset an SoC in the case of main program exception.

3.17.1 WDT Reset

The WDT Reset example project enables resetting an SoC at certain intervals with a WDT.

The source code and project file of the WDT Reset example are in SDK_Folder\projects\peripheral\wdt\wdt_reset, and project file is in the Keil_5 folder.

3.17.1.1 Code Interpretation

Figure 3-42 shows the workflow of a WDT Reset example project.

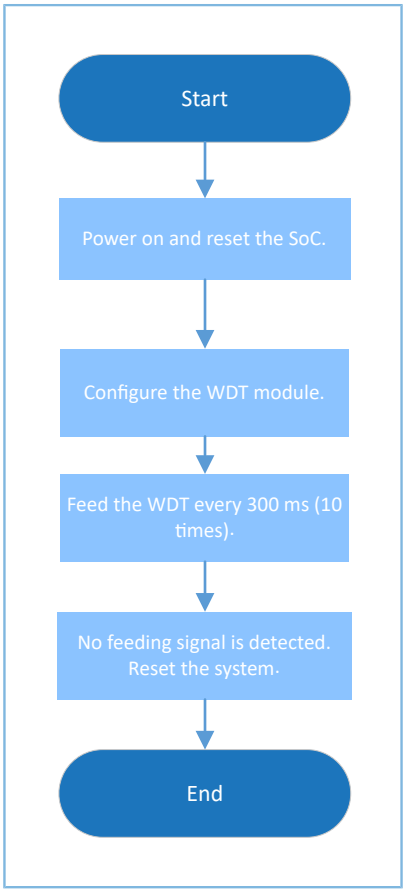

Figure 3-42 WDT Reset example workflow

1. Configure WDT module.

```
g_wdt_handle.init.counter    = SystemCoreClock;
g_wdt_handle.init.reset_mode = WDT_RESET_ENABLE;
hal_wdt_init(&g_wdt_handle);
```

   - counter: for WDT timing, set to 1 second
   - reset_mode: to enable/disable SoC reset, which is set to WDT_RESET_ENABLE

2. Feed the WDT by calling hal_wdt_refresh(). The code snippet is as follows:

```
hal_wdt_refresh(&g_wdt_handle);
```

3.17.1.2 Test and Verification

1. Download *wdt_reset_sk_r2_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the debugging information printed every 300 milliseconds in the **Receive Data** pane on GRUart. The SoC restarts after printing for 10 times.

3.18 COMP

A comparator (COMP) compares the current/voltage levels from two inputs and provides result in outputs. When the voltage is higher than the threshold, interrupt will be triggered. It is widely used in analog-to-digital converters (ADCs).

3.18.1 COMP MSIO

A COMP MSIO example project enables comparing input voltages of two MSIO pins.

The source code and project file of the COMP MSIO example are in SDK_Folder\projects\peripheral\comp\comp_msio_msio, and project file is in the Keil_5 folder.

3.18.1.1 Code Interpretation

Figure 3-43 shows the workflow of a COMP MSIO example.

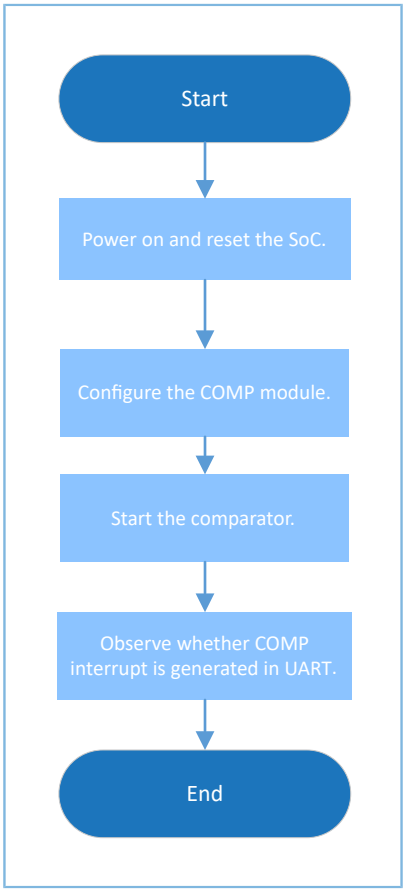

Figure 3-43 COMP MSIO example workflow

1. Configure the COMP module.

```
g_comp_handle.init.input_source = COMP_INPUT_SRC_IO0;
g_comp_handle.init.ref_source   = COMP_REF_SRC_IO1;
g_comp_handle.init.ref_value    = 0;
hal_comp_deinit(&g_comp_handle);
hal_comp_init(&g_comp_handle);

msio_config.pin = COMP_INPUT_PIN | COMP_VREF_PIN;
```



```
msio_config.direction = MSIO_DIRECTION_INPUT;
msio_config.pull = MSIO_PULLUP;
msio_config.mode = MSIO_MODE_ANALOG;
hal_msio_init(&msio_config);
```

- input_source: COMP input pins. Options: COMP_INPUT_SRC_IO0 – COMP_INPUT_SRC_IO4, corresponding pins: MSIO0 – MSIO4
- ref_source: reference source. Options: COMP_REF_SRC_IO0 – COMP_REF_SRC_VREF, corresponding pins: MSIO0 – MSIO4 (external reference source); VBAT (battery voltage), VREF (internal reference source)
- ref_value: reference input voltage of a COMP. The COMP example project uses an external power COMP_REF_SRC_IO1 as voltage input, which is not a certain value. The value is not required to be set, or set as 0 (default value).
- Configure COMP_INPUT_PIN (MSIO 0) and COMP_VREF_PIN (MSIO 1) as analog input pins.

2. Call hal_comp_start() to start the COMP in interrupt mode. Comparison results are returned by calling hal_comp_trigger_callback(). Users can customize executable operations in this callback function. The code snippet is as follows:

```
void hal_comp_trigger_callback(comp_handle_t *p_comp)
{
    printf("Comp is triggered.\r\n");
}

hal_comp_start(&g_comp_handle);
```

3.18.1.2 Test and Verification

1. Download *comp_msio_msio_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the comparison results from the COMP in the **Receive Data** pane on GRUart.

3.18.2 COMP VBAT

A COMP VBAT example project enables to compare a battery voltage and an input voltage of MSIO pins.

The source code and project file of the COMP VBAT example are in SDK_Folder\projects\peripheral\comp\comp_msio_vbat, and project file is in the Keil_5 folder.

3.18.2.1 Code Interpretation

Figure 3-44 shows the workflow of a COMP VBAT example project.

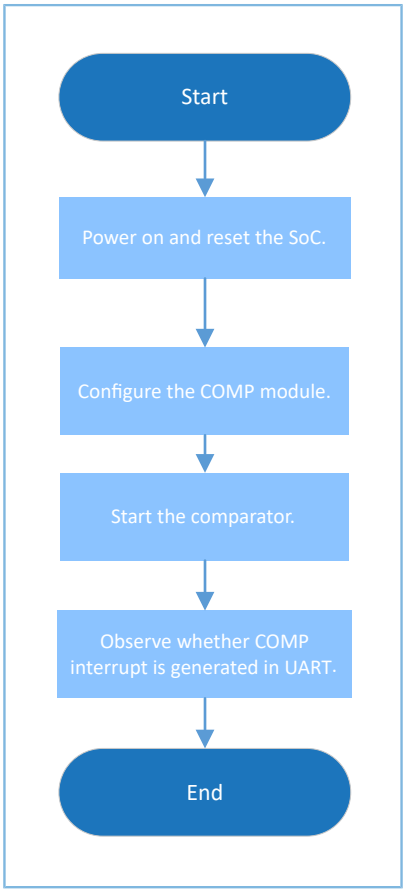

Figure 3-44 COMP VBAT example workflow

1. Configure the COMP module.

```
g_comp_handle.init.input_source = COMP_INPUT_SRC_IO0;
g_comp_handle.init.ref_source   = COMP_REF_SRC_VBAT;
g_comp_handle.init.ref_value    = 5;  // Reference = ((ref_value + 1) / 10) * VBATT
hal_comp_deinit(&g_comp_handle);
hal_comp_init(&g_comp_handle);

msio_config.pin = COMP_INPUT_PIN;
msio_config.direction = MSIO_DIRECTION_INPUT;
msio_config.pull = MSIO_PULLUP;
msio_config.mode = MSIO_MODE_ANALOG;
hal_msio_init(&msio_config);
hal_timer_base_init(&g_tim1_handle);
```

   - For more information about input_source and ref_source, see “Section 3.18.1 COMP MSIO”.
   - ref_value: reference voltage of the COMP. Range: 0 to 7. When the value is 0, the comparator fails to meet specifications.
   - Configure COMP_INPUT_PIN (MSIO 0) as the analog input pin.

2. Call hal_comp_start() to start the COMP in interrupt mode. Comparison results are returned by calling hal_comp_trigger_callback(). Users can customize executable operations in this callback function. The code snippet is as follows:

```
void hal_comp_trigger_callback(comp_handle_t *p_comp)
{
    printf("Comp is triggered.\r\n");
}

hal_comp_start(&g_comp_handle);
```

3.18.2.2 Test and Verification

1. Download *comp_msio_msio_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in "Section 2.4 Serial Port Settings".
3. You can view the comparison results from the COMP in the **Receive Data** pane on GRUart.

3.18.3 COMP Vref

The COMP Vref example project enables to compare the internal reference voltage with external voltage from MSIO pins.

The source code and project file of the COMP Vref example are in SDK_Folder\projects\peripheral\comp\comp_msio_vref, and project file is in the Keil_5 folder.

3.18.3.1 Code Interpretation

Figure 3-45 shows the workflow of a COMP Vref example project.

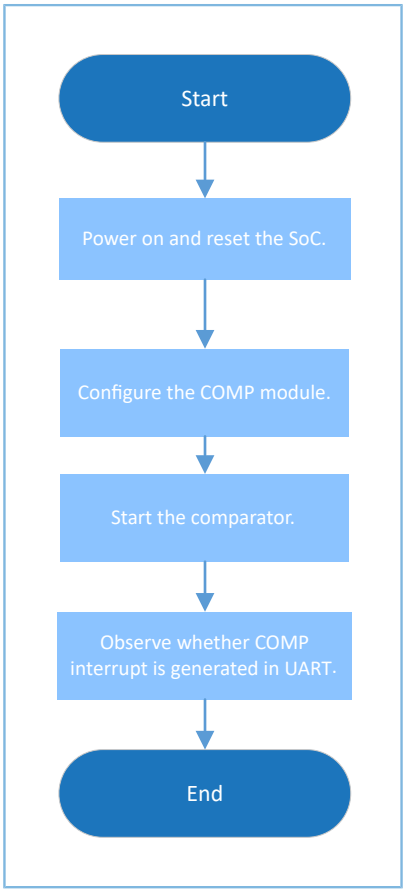

Figure 3-45 COMP Vref example workflow

1. Configure the COMP module.

```
g_comp_handle.init.input_source = COMP_INPUT_SRC_IO0;
g_comp_handle.init.ref_source   = COMP_REF_SRC_VREF;
g_comp_handle.init.ref_value    = 30;  // Reference = 30mv * ref_value
hal_comp_deinit(&g_comp_handle);
hal_comp_init(&g_comp_handle);
```

   - For more information about input_source and ref_source, see “Section 3.18.1 COMP MSIO”.
   - ref_value: reference voltage of the COMP. For this COMP example, the voltage range is from 0 to 63. When the voltage is 0, the comparator fails to meet specifications.

2. Call hal_comp_start() to start the COMP in interrupt mode. Comparison results are returned by calling hal_comp_trigger_callback(). Users can customize executable operations in this callback function. The code snippet is as follows:

```
void hal_comp_trigger_callback(comp_handle_t *p_comp)
{
    printf("Comp is triggered.\r\n");
}

hal_comp_start(&g_comp_handle);
```

3.18.3.2 Test and Verification

1. Download *comp_msio_vref_fw.bin* to a GR5515 SK Board through GProgrammer.
2. Connect the serial port of the board to the PC, start GRUart, and configure the serial port on the GRUart as described in “Section 2.4 Serial Port Settings”.
3. You can view the comparison results from the COMP in the **Receive Data** pane on GRUart.

4 FAQ

This chapter describes possible problems, reasons, and solutions during verification and application of the peripheral examples.

4.1 FAQ about SPI

- Description
  1. Exception in data TX via DMA occurs when configuring chip select (CS) pins through hardware control.
  2. The transfer speed of HAL SPI cannot reach 32 MHz.
- Analysis
  1. When the CS pins are configured through hardware control, once TX FIFO is empty, the hardware determines that data TX is completed, and therefore automatically pulls up CS signals. The reason may be that SPI data TX is interrupted for tasks with higher priority, and therefore the TX FIFO is not entered in time.
  2. At HAL, CS pins are pulled up automatically when TX FIFO is empty, for which the clock cannot provide the 32 MHz in full for data transfer. The transfer speed varies depending on the transfer mode: DMA > polling > interrupt (from high to low). When sending 8-bit data (mirroring), the speed can reach 16 MHz in DMA mode, 4 MHz – 8 MHz in polling mode, and even lower in interrupt mode.
- Solution
  1. Configure the CS pins as GPIOs and enable the pins with support from Pin Mux.
  2. Use SPI clocks lower than 8 MHz or use LL interfaces to operate the registers.

4.2 FAQ about QSPI

- Description
  1. Data TX exception occurs when CS pins are used to configure hardware control.
- Analysis
  1. When the CS pins are configured through hardware control, once TX FIFO is empty, the hardware determines that data TX is completed, and therefore automatically pulls up CS signals. The reason may be that QSPI data TX is interrupted for tasks with higher priority, and therefore the TX FIFO is not entered in time.
  2. The API hal_qspi_transmit() does not support QSPI transfer with speed at or higher than 8 MHz.
- Solution
  1. Configure the CS pins as GPIOs and enable the pins with support from Pin Mux.
  2. Use QSPI in DMA mode.

4.3 FAQ about ADC

- Description

  1. ADC adopts single-ended input and calculates sampling rates with formulas. The calculation results turn out to be wrong.
  2. Wrong reference source is selected, causing errors in test result.
  3. After initializing an ADC, enable the ADC immediately to get sampling values. The first values are wrong.

- Analysis

  1. The check formulas are not used correctly.
  2. The selected reference source is wrong. At present, MSIO 4 and VBAT cannot be used as the input reference source.
  3. An ADC is not in the best operating state immediately after initialization. If you collect data at that time, exception will occur.

- Solution

  1. Use the correct check formula:
     - Single-ended (internal): V = (ADC - offset) / ((-1) * slope)

       The offset and slope refer to the offset correction and slope value respectively stored in an SoC at corresponding internal reference voltage.
     - Single-ended (external): V = (ADC - offset) / (((-1) * slope * 1.0f) / vref)

       The offset and slope refer to the offset correction and slope value respectively stored in an SoC at corresponding external reference voltage (1.0 V).
     - Differential (internal): V = 2 * (offset – ADC) / ((-1) * slope)

       The offset refers to the corrected code value in the case of P = N; the slope refers to the slope value stored in an SoC at corresponding internal reference voltage.
     - Differential (external): V = 2 * (offset – 2 * ADC) / (((-1) * slope * 1.0f) / vref)

       The offset refers to the corrected code value in the case of P = N; the slope refers to the slope value stored in an SoC at corresponding external reference voltage (1.0 V).
  2. Use the reference source specified in the header files as input.
  3. After hal_adc_init() is called, the first data entries should not be used when running hal_gr551x_adc_voltage_intern() or hal_adc_start_dma() (the exact number of data entries to be discarded depends on the sampling rate of an ADC).

4.4 FAQ for PWM

- Description

  The duty ratios of three channels of PWM are independent and can be adjusted. However, when the three channels are set at different frequencies, the channels are invalid or run into problems.

- Analysis

  The frequencies of the channels shall be the same. Adjustment fails because frequencies are different.

- Solution

  Set the frequencies of the three PWM channels to the same value.