GR5526 Display Refresh Module Guide

Version: 1.0

Release Date: 2023-01-10

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: Floor 12-13, Phase B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 Zip Code: 518000

Website: www.goodix.com

Preface

Purpose

This document introduces the display refresh module in GR5526 System-on-Chips (SoCs) which is applied to wearable devices, to help users quickly get started with the functionalities and features of the module and accelerate the development and performance optimization of wearable devices.

Audience

This document is intended for:

- GR5526 user
- GR5526 developer
- GR5526 tester
- Hobbyist developer
- Technical writer

Release Notes

This document is the initial release of *GR5526 Display Refresh Module Guide*, corresponding to GR5526 SoC series.

Revision History

Version	Date	Description
1.0	2023-01-10	Initial release

Contents

1 Overview

GR5526 System-on-Chip (SoC) series is provided in two versions (standard version and GPU version) based on whether the GPU module is embedded and two packages (BGA83/QFN68). Refer to the table below for details.

Table 1-1 GR5526 series

GR5526 Series	RAM	SiP Flash	SiP PSRAM	I/O Number	Package (mm)	GPU Supported
GR5526VGBIP	512 KB	1 MB	8 MB	50	BGA83 (4.3 x 4.3 x 0.96)	Yes
GR5526VGBI	512 KB	1 MB	N/A	50	BGA83 (4.3 x 4.3 x 0.96)	N/A
GR5526RGNIP	512 KB	1 MB	8 MB	48	QFN68 (7.0 x 7.0 x 0.85)	Yes
GR5526RGNI	512 KB	1 MB	N/A	48	QFN68 (7.0 x 7.0 x 0.85)	N/A

1.1 Display Refresh Elements

GR5526 provides rich sets of display refresh elements, as listed below.

Table 1-2 GR5526 display refresh element summary 1

Version	System Main Frequency	X-Flash	External QSPI Frequency	SRAM	QSPI0	QSPI1
Standard version	96 MHz	1 MB	48 MHz	512 KB	• NOR Flash (read in XIP mode) • NAND Flash (access in register mode) • Q-PSRAM (read/write in XIP mode) • Display • DMA0 (register mode) • DMA0/DMA1 (XIP mode) • Dynamic adaptive endian • Multiple types of static endian • 1-wire/2-wire/4-wire mode	• NOR Flash (read in XIP mode) • NAND Flash (access in register mode) • Q-PSRAM (read/write in XIP mode) • Display • DMA0/DMA1 (register mode) • DMA0/DMA1 (XIP mode) • Dynamic adaptive endian • Multiple types of static endian • 1-wire/2-wire/4-wire mode
GPU version						

Table 1-3 GR5526 display refresh element summary 2

Version	QSPI2	DMA	SPIM	DSPI	OSPI DDR PSRAM	GPU	DC
Standard version	• NOR Flash (read in XIP mode) • NAND Flash (access in register mode) • Q-PSRAM (read/write in XIP mode) • Display	• 2 instances • Linked List • Scatter • Gather	48 MHz	48 MHz	N/A	N/A	N/A
GPU version					48 MHz	96 MHz	48 MHz

Version	QSPI2	DMA	SPIM	DSPI	OSPI DDR PSRAM	GPU	DC
	• DMA1 (register mode) • DMA0/DMA1 (XIP mode) • Dynamic adaptive endian • Multiple types of static endian • 1-wire/2-wire/4-wire mode	• Large FIFO depth					

Major display refresh elements are categorized as follows:

1. Computing power
   (1). CPU
   (2). GPU (SoC with GPU)
2. Persistent storage of materials/texture data
   (1). NOR Flash
   (2). NAND Flash
   (3). XQSPI Flash (remaining code space, part of which can be used to store data)
3. Computation and buffer space
   (1). 512 KB on-chip SRAM
   (2). External QSPI PSRAM up to 512 Mbit
   (3). Embedded 64 Mbit OSPI DDR PSRAM (SoC with GPU)
4. Data transmission enhancement/Frame rate acceleration
   (1). DMA0/DMA1
5. Display interfaces (covering MIPI DBI Type-C interface)
   (1). QSPI interface
   (2). SPI master interface
   (3). Display SPI interface
   (4). Display Controller (DC) interface (SoC with GPU)
6. Graphic effect enhancement
   (1). GPU (SoC with GPU)
   (2). DMA

1.2 Display Refresh Models

According to the market positioning of different wearable products, multitude of display refresh models can be built with various combinations of display refresh elements.

This document focuses on typical features and applications of DMA, QSPI, and Display Controller modules. For introduction to each module, refer to *GR5526 Datasheet*; for details of the GPU module, refer to *GR5526 GPU Developer Guide*.

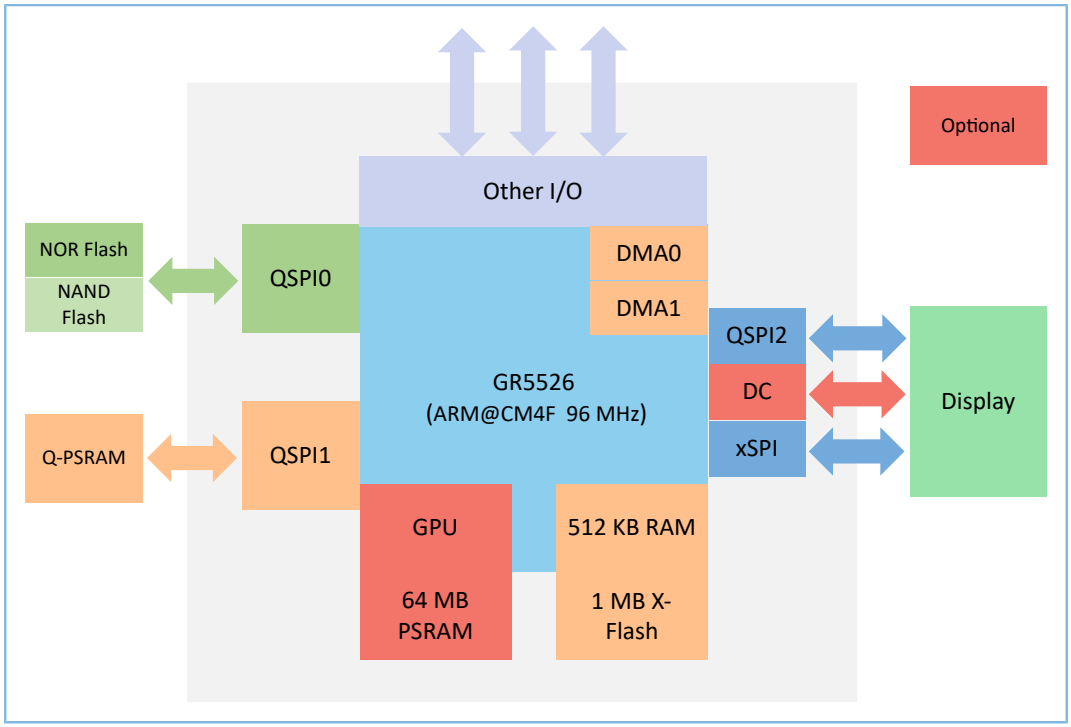

Figure 1-1 Typical display refresh block diagram of GR5526

2 Typical DMA Features

For details of Direct Memory Access (DMA), refer to DMA-related contents in *GR5526 Datasheet*.

2.1 Typical DMA Applications

2.1.1 DMA Base Data Transmission

DMA enables GR5526 to transmit data from memory to memory, memory to peripheral, peripheral to memory, and peripheral to peripheral.

In a DMA base data transmission, up to 4095 beats of data can be transmitted at a time. “Beat” refers to the bit width of the data transmitted using DMA.

- If the data bit width is in bytes, up to 4095 bytes of data can be transmitted in a single DMA transmission.
- If the data bit width is in half-words, up to 4095 x 2 bytes of data can be transmitted in a single DMA transmission.
- If the data bit width is in words, up to 4095 x 4 bytes of data can be transmitted in a single DMA transmission.

If more than 4095 beats of data are transmitted in a single DMA base data transmission, an error message will pop up, indicating that the DMA transmission is interrupted.

Note:

Address alignment shall be ensured in DMA transmission.

2.1.2 DMA Chain Data Transmission

To improve the DMA transmission capability in the following scenarios, GR5526 SoCs introduce DMA chain data transmission, which means multiple data blocks can be connected using a pointer linked list, so that all these data can be transmitted in a single DMA transmission cycle.

- Transmit more than 4095 beats of data in a single transmission.
- Transmit data from a discontinuous address space in a single transmission cycle.
- Use different transmission configurations to transmit data in a single transmission cycle.

To be specific: Manage all to-be-transmitted data blocks using a pointer linked list. After a data block is sent, the next block is automatically loaded according to the .next pointer until the .next pointer becomes empty.

As shown below, each link node mainly contains DMA transmission configuration information of the current node, node data block, and information of the pointer pointing to the next link node, until the last transmission link node becomes null. These link nodes will be implemented in a single DMA transmission. Data at each node shall not be more than 4095 beats.

DMA chain data transmission can be widely used in such scenarios as big data block transmission, discontinuous data transmission, and display refresh.

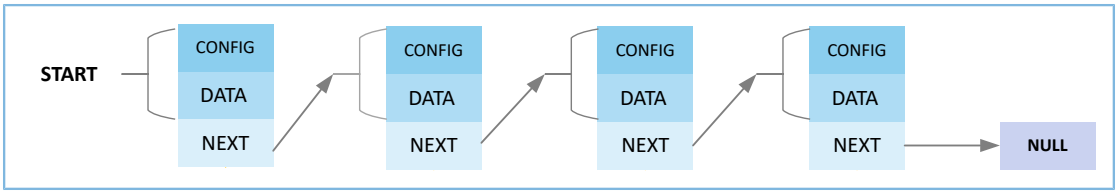

Figure 2-1 DMA chain data transmission

2.1.3 DMA Scatter Transmission

In a DMA scatter transmission, data in a continuous address space is scattered to a discontinuous address space based on certain rules.

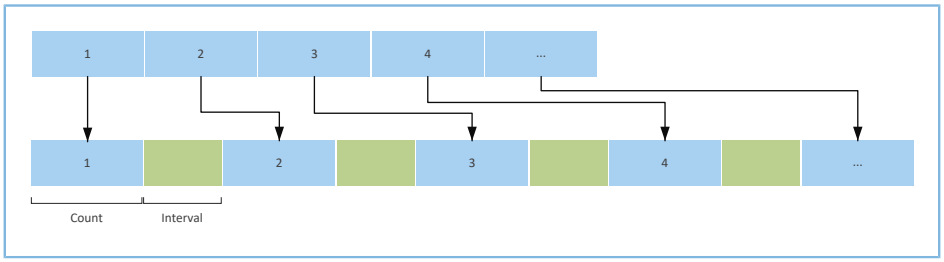

Figure 2-2 DMA scatter transmission schematic

1. The continuous data is divided equally into several data blocks, and the length of the last data block can be different.
2. The data volume contained in each data block is marked as “count” (in beats).
3. The address interval between data blocks is marked as “interval” (in beats).

Example: To transmit 1000 bytes of data (count: 4; interval: 2) through DMA scatter transmission:

- If the transmission width is 8 bits, then the data is 1000 beats in total, each data block is 4 bytes, and the address interval is 2 bytes.
- If the transmission width is 16 bits, then the data is 500 beats in total, each data block is 8 bytes, and the address interval is 4 bytes.
- If the transmission width is 32 bits, then the data is 200 beats in total, each data block is 16 bytes, and the address interval is 8 bytes.

2.1.4 DMA Gather Transmission

In a DMA gather transmission, data in a discontinuous address space is gathered in a continuous address space. DMA gather transmission can be regarded as a reverse process of DMA scatter transmission.

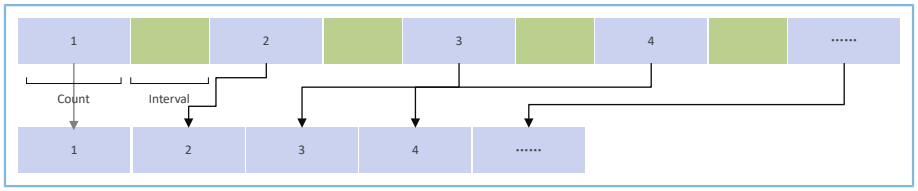

Figure 2-3 DMA gather transmission schematic

1. Several data blocks of equal length are evenly arranged at the same address interval, and the length of the last data block can be different.
2. The data volume contained in each data block is marked as “count” (in beats).
3. The address interval between data blocks is marked as “interval” (in beats).

Example: To transmit 1000 bytes of data (count: 4; interval: 2) through DMA gather transmission:

- If the transmission width is 8 bits, then the data is 1000 beats in total, each data block is 4 bytes, and the address interval is 2 bytes.
- If the transmission width is 16 bits, then the data is 500 beats in total, each data block is 8 bytes, and the address interval is 4 bytes.
- If the transmission width is 32 bits, then the data is 200 beats in total, each data block is 16 bytes, and the address interval is 8 bytes.

2.1.5 DMA Scatter and Gather Transmission

The DMA scatter and gather transmission can be enabled simultaneously, to transmit the data in a discontinuous address space to another discontinuous address space.

2.2 Features of DMA Transmission Channel

GR5526 provides two DMA instances for users: DMA0 and DMA1. There are six DMA channels in total. The FIFO depth is 32 (channel 0) or 4 (channels 1–5).

Channel 0 features large FIFO depth, so it can cache more data in DMA transmission, improving DMA transmission throughput. Therefore, in wearable applications, it is recommended to allocate channel 0 of DMA0/DMA1 to peripherals with high throughput in such scenarios as Flash access, PSRAM access, bulk memory block transfer, and display.

3 Typical QSPI Features

For details of QSPI, refer to QSPI-related contents in *GR5526 Datasheet*.

3.1 QSPI Working Modes

GR5526 QSPI works in three modes according to the number of data cables in use and the sequence:

- 1-wire SPI mode: SCLK, CS#, MOSI, and MISO signals are involved.
- 2-wire Dual SPI mode: SCLK, CS#, IO0, and IO1 signals are involved.
- 4-wire Quad SPI mode: SCLK, CS#, IO0, IO1, IO2, and IO3 signals are involved.

GR5526 QSPI works in two modes according to the specific peripheral access methods:

- Register mode: After initialization of the QSPI module and peripherals, QSPI reads data from/writes data to peripherals by controlling registers. These control operations are encapsulated as functions in SDK by functionalities, and specific accesses are implemented by calling corresponding functions (arrays).
- Memory mapped mode: Also called “XIP mode”. After initialization of the QSPI module and peripherals, the memory space of peripherals is mapped to the bus address space of the system, and then QSPI accesses peripherals by bus addressing.

Generally, read/write access to any peripheral that supports the QSPI sequence protocol can be implemented in register mode, such as NOR Flash, NAND Flash, PSRAM, display, and other peripherals with QSPI interface and supporting QSPI sequence.

QSPI NOR Flash and QSPI PSRAM can also be accessed in memory mapped mode. Due to different access characteristics, QSPI NOR Flash only supports read operations in memory mapped mode, whereas QSPI PSRAM supports both read and write operations in this mode.

Note:

Generally, storage devices supporting memory mapped mode can only work in Dual SPI/Quad SPI mode. Additionally, to ensure higher access efficiency, it is recommended to work in Quad SPI mode.

The table below lists the access methods supported by common wearable peripherals with recommendations as follows:

1. For specific modes supported by a peripheral in practice, you can refer to the datasheet of the peripheral.
2. It is recommended to adopt the memory mapped mode if supported by peripherals, to make the program code used to read data from the storage device more concise and access more efficient.

Table 3-1 GR5526 QSPI working modes supported by wearable peripherals

Working Mode		NOR Flash		NAND Flash		QSPI PSRAM		Display/LCD	
		Read	Write	Read	Write	Read	Write	Read	Write
Register Mode	SPI	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
	Dual SPI	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Working Mode		NOR Flash		NAND Flash		QSPI PSRAM		Display/LCD	
		Read	Write	Read	Write	Read	Write	Read	Write
	Quad SPI	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Memory Mapped Mode	SPI	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
	Dual SPI	Yes	N/A	N/A	N/A	Yes	Yes	N/A	N/A
	Quad SPI	Yes	N/A	N/A	N/A	Yes	Yes	N/A	N/A

3.2 QSPI Data Endian

3.2.1 Data Endian in Write Operations

When the same memory data is written to a peripheral through QSPI with different bus access widths using CPU or DMA, different types of byte endian will be obtained, as shown below.

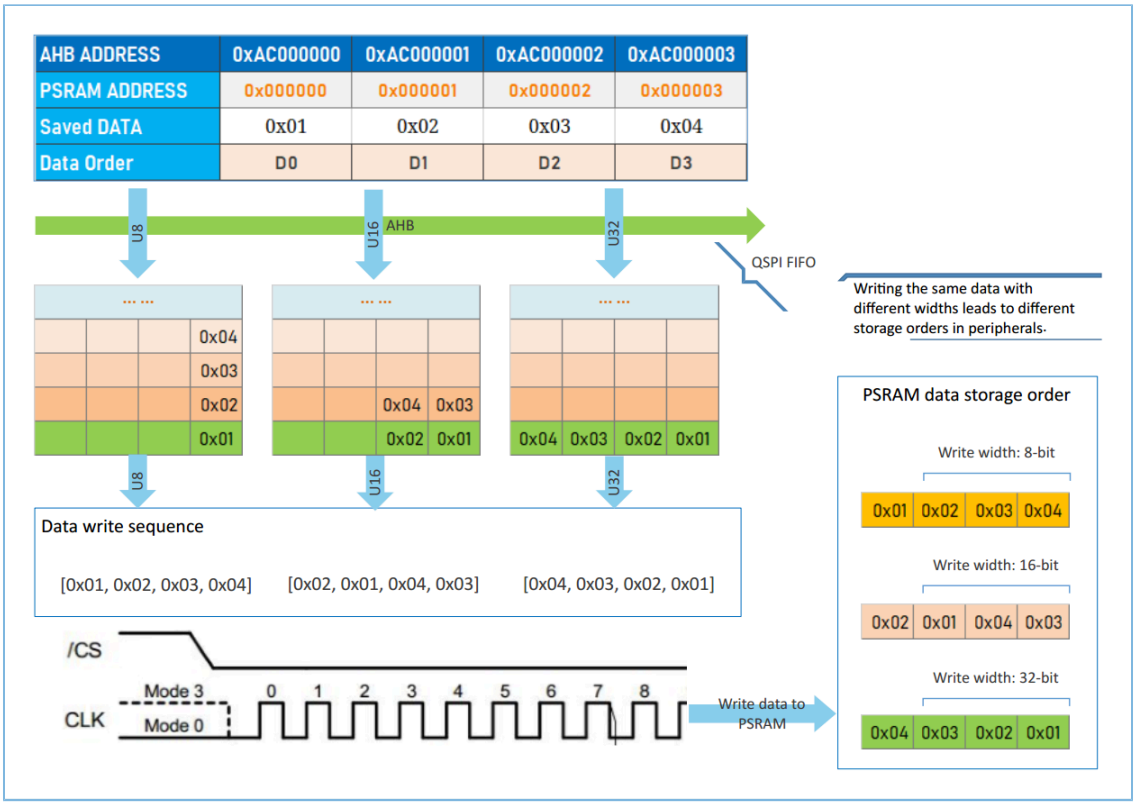

Figure 3-1 Default write process of Synopsys QSPI controller

The endian conversion process is as follows:

1. Send the array {0x01, 0x02, 0x03, 0x04} through QSPI.
2. When the data enters the QSPI FIFO queue:
   (1). If the data is to be written by bytes, 4 FIFO depths will be occupied. The array is arranged as 0x01, 0x02, 0x03, and 0x04 in order.

(2). If the data is to be written by half-words, the data is converted into 0x0201 and 0x0403 on the bus, and 2 FIFO depths will be occupied in QSPI. The array is arranged as {0x02,0x01} and {0x04,0x03} in order.

(3). If the data is to be written by words, the data is converted into 0x04030201 on the bus, and 1 FIFO depth will be occupied in QSPI. The array is arranged as {0x04, 0x03, 0x02, 0x01}.

3. QSPI FIFO outputs data through data cables from MSB to LSB, so the order of the data in data cables is shown below:

(1). If the data is to be written by bytes, the data in the sequence line is displayed as 0x01,0x02,0x03,0x04.

(2). If the data is to be written by half-words, the data in the sequence line is displayed as 0x02,0x01,0x04,0x03.

(3). If the data is to be written by words, the data in the sequence line is displayed as 0x04,0x03,0x02,0x01.

4. Peripherals store/process the data according to the data endian.

Note:

By default, data write operations in both register mode and memory mapped mode follow the above endian conversion rules.

3.2.2 Data Endian in Read Operations

Read operations are a reverse process when compared with write operations. Different read access widths are adopted for the data in the address space of peripherals, to obtain data with different types of byte endian, as shown below.

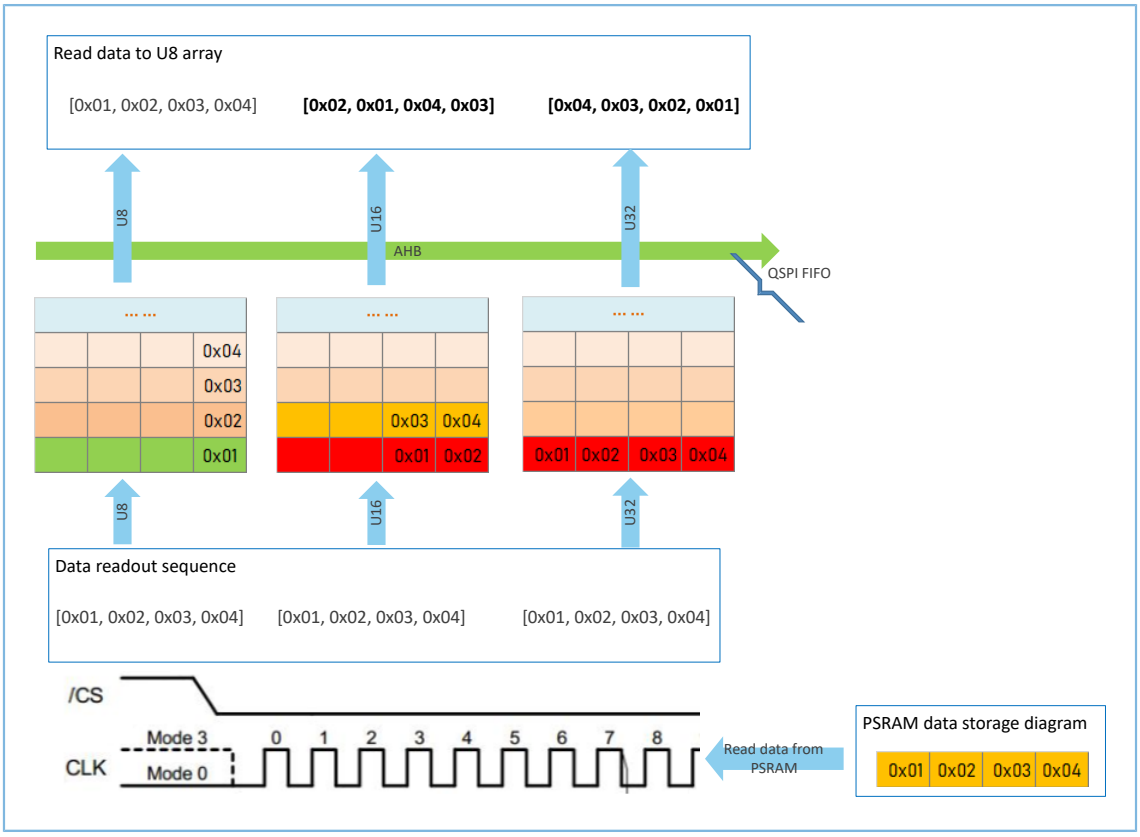

Figure 3-2 Data endian in read operations

The endian conversion process is as follows:

- If the data width is in bytes, read {0x01,0x02,0x03,0x04} in the peripheral space to the memory space as {0x01,0x02,0x03,0x04}.
- If the data width is in half-words, read {0x01,0x02,0x03,0x04} in the peripheral space to the memory space as {0x02,0x01,0x04,0x03}.
- If the data width is in words, read {0x01,0x02,0x03,0x04} in the peripheral space to the memory space as {0x04,0x03,0x02,0x01}.

By default, data read operations in both register mode and memory mapped mode follow the above endian conversion rules. To obtain the same byte endian, access widths of read and write operations shall be kept the same.

However, in wearable product applications, the minimum access unit of resources such as images and fonts may be byte, half-word, word, or any combination of them due to different color formats (such as RGBA8888 and ARGB8888) and color depths (such as RGBA8888 and RGB565). Therefore, processing of byte endian is required during software access. If endian is adjusted by software, a lot of CPU computing power will be consumed. GR5526 SoCs are designed with endian modes supporting various access scenarios.

3.2.3 Read Rule for Static Data Endian in Memory Mapped Mode

The static data endian refers to a data endian in which data is output correspondingly according to different register configurations (hereinafter referred to as “static endian rule”).

- The static endian rule applies only when QSPI works in memory mapped mode.
- The static endian rule is typically used to read data from QSPI NOR Flash.

According to the static endian rule, for a fixed data storage sequence, you can configure registers to output different types of data read endian. The table below lists data stored in peripheral memory.

Table 3-2 Peripheral data to be read

Peripheral Address	0x000000	0x000001	0x000002	0x000003
Stored Data	0x01	0x02	0x03	0x04

Store the data 0x01, 0x02, 0x03, and 0x04 in the peripheral address space 0x000000–0x000003 in order.

The table below lists the data obtained when different data types are adopted to access the above peripheral data with different endian rules.

Table 3-3 Static endian rule for QSPI read operations

Access Type	Byte				Half-word		Word
Access Address	0xAC000000	0xAC000001	0xAC000002	0xAC000003	0xAC000000	0xAC000002	0xAC000000
Endian Mode 0	0x01	0x02	0x03	0x04	0x0102	0x0304	0x01020304
Endian Mode 1	0x01	0x02	0x03	0x04	0x0201	0x0403	0x02010403
Endian Mode 2	0x01	0x02	0x03	0x04	0x0201	0x0403	0x04030201

Descriptions about the static endian rule:

- Assume that the mapping address of the peripheral address 0x000000–0x000003 in the bus address space is 0xAC000000–0xAC000003.
- When static endian mode 0 is configured:
  - When 0xAC000000 is accessed in bytes (corresponding to uint8_t * in C language), 0x01 is returned.
  - When 0xAC000000 is accessed in half-words (corresponding to uint16_t * in C language), 0x0102 is returned.
  - When 0xAC000000 is accessed in words (corresponding to uint32_t * in C language), 0x01020304 is returned.
- When static endian mode 1 is configured:
  - When 0xAC000000 is accessed in bytes (corresponding to uint8_t * in C language), 0x01 is returned.
  - When 0xAC000000 is accessed in half-words (corresponding to uint16_t * in C language), 0x0201 is returned.
  - When 0xAC000000 is accessed in words (corresponding to uint32_t * in C language), 0x02010403 is returned.
- When static endian mode 2 is configured:
  - When 0xAC000000 is accessed in bytes (corresponding to uint8_t * in C language), 0x01 is returned.

- When 0xAC000000 is accessed in half-words (corresponding to uint16_t * in C language), 0x0201 is returned.
- When 0xAC000000 is accessed in words (corresponding to uint32_t * in C language), 0x04030201 is returned.

You can set a proper endian rule on demand to ensure optimal efficiency of QSPI.

Set the function for the static endian rule: app_qspi_mmap_set_endian_mode(). For details, refer to "Section 3.3.3.4 app_qspi_mmap_set_endian_mode".

3.2.4 Read Rule for Dynamic Data Endian in Memory Mapped Mode

The dynamic data endian refers to a data read endian in which data is output in memory mapped mode according to different data access types (hereinafter referred to as "dynamic endian rule").

- The dynamic endian rule applies only when QSPI works in memory mapped mode.
- The dynamic endian rule is typically used to read data from/write data to QSPI PSRAM.
- The dynamic endian rule has a higher priority than the static endian rule. If both dynamic and static endian rules are enabled, the system responds to the dynamic endian rule only.

According to the dynamic endian rule, when a read/write access occurs, the behavior of data entering FIFO can be modified to automatically adapt to mixed access to different types of data.

The figure below shows the behavior of different types of data entering QSPI FIFO during read and write operations after the dynamic endian rule is enabled. The dynamic endian rule helps ensure write consistency for different types of base data.

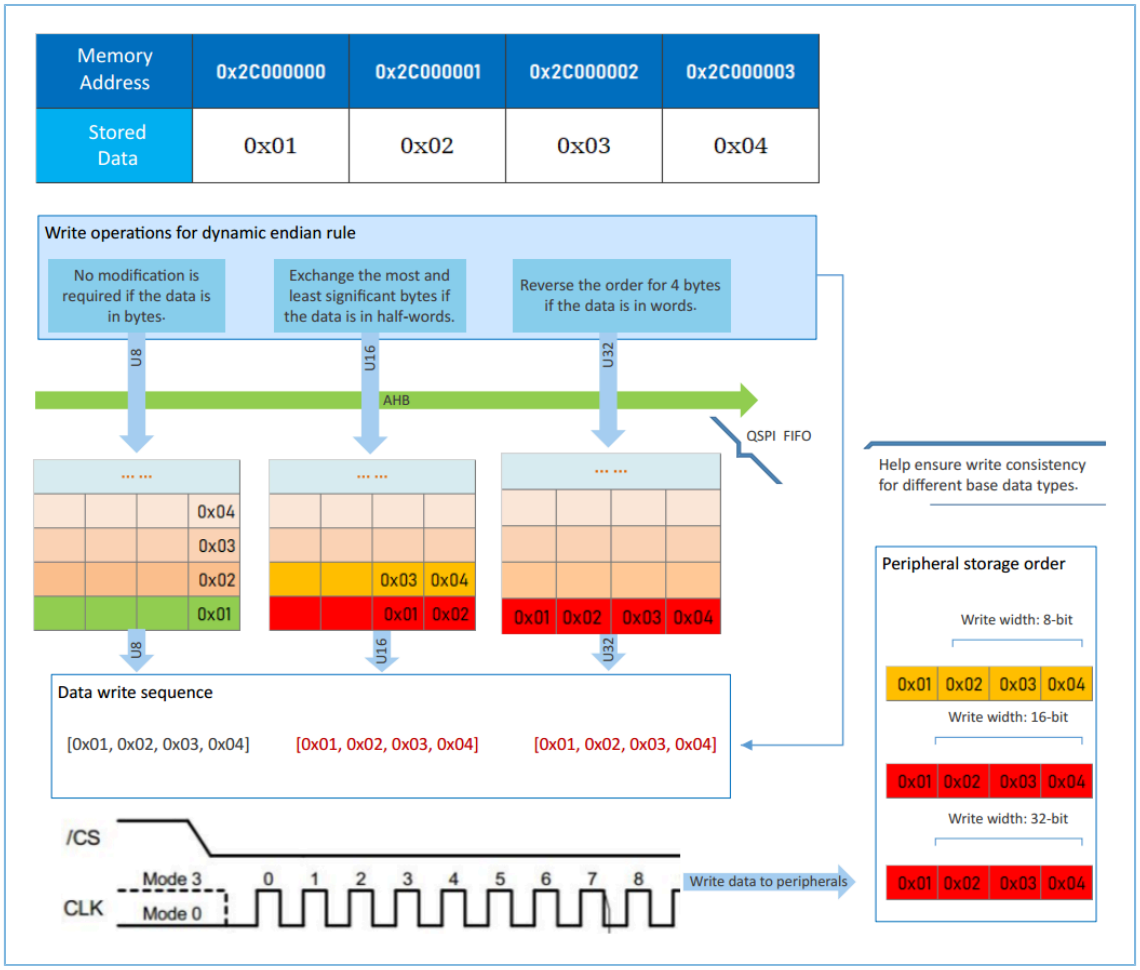

Figure 3-3 Behavior of different types of data entering QSPI FIFO for write operations

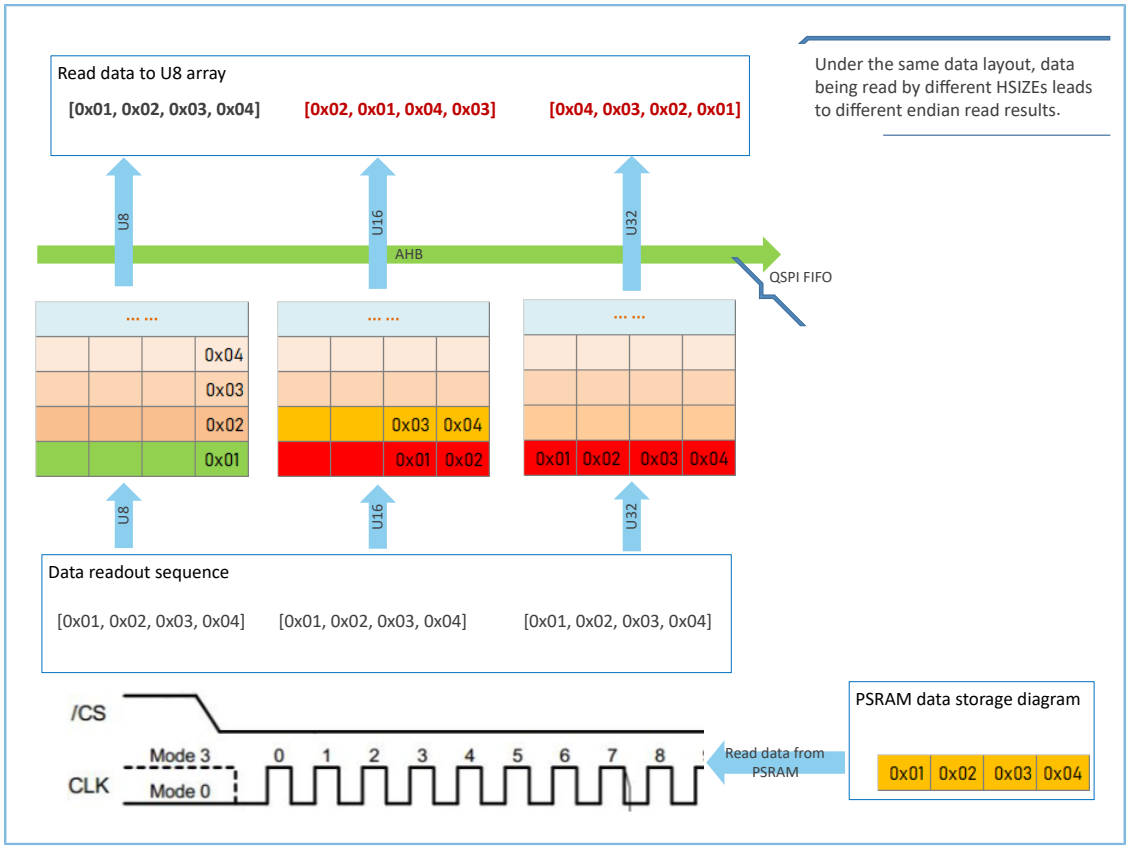

Figure 3-4 Behavior of different types of data entering QSPI FIFO for read operations

When QSPI PSRAM initializes the memory mapped mode, the GR5526 driver automatically enables the dynamic endian rule, to make PSRAM access consistent with SRAM access.

3.3 QSPI Driver Operational Instructions

3.3.1 Common Functions

3.3.1.1 app_qspi_init

Table 3-4 app_qspi_init API

Function Prototype	uint16_t app_qspi_init(app_qspi_params_t *p_params, app_qspi_evt_handler_t evt_handler)
Function Description	Initialize the QSPI module.
Parameter	• p_params: initialization parameters • evt_handler: Register the function for notifying asynchronous API callback events.
Return Value	Refer to APP_DRV_ERR_CODE in the source code.
Remarks	

app_qspi_params_t definition and members are detailed as follows.

```
typedef struct
```

```
{
    app_qspi_id_t          id;
    app_qspi_pin_cfg_t     pin_cfg;
    app_qspi_dma_cfg_t     dma_cfg;
    qspi_init_t            init;
} app_qspi_params_t;
```

id

- APP_QSPI_ID_0: QSPI0
- APP_QSPI_ID_1: QSPI1
- APP_QSPI_ID_2: QSPI2

pin_cfg

- cs: Configure the CS pin.
  - type: pin type
  - mux: the way of I/O multiplexing
  - pin: pin number
  - mode: Configure the I/O mode.
  - pull: Activate pull-up or pull-down resistors.
  - enable: Enable/Disable the CS pin.
- clk: Configure the clock pin.
- io_0: QSPI IO0 (SPI MOSI)
- io_1: QSPI IO1 (SPI MISO)
- io_2: QSPI IO2
- io_3: QSPI IO3

By default, three groups of QSPI pin configurations are provided by the driver and are defined in g_qspi_pin_groups, which can be used directly or redefined as needed.

dma_cfg

- dma_instance: Allocate a DMA instance to the selected QSPI module (ID) when DMA transfer is adopted, with DMA0 for QSPI0, DMA0/DMA1 for QSPI1, and DMA2 for QSPI2.
- dma_channel: Allocate a channel to the DMA instance when DMA transfer is adopted.
- wait_timeout_ms: Set the timeout period (in ms) for polling APIs.
- extend: Reserved bits, not in use

init

- clock_prescaler: QSPI clock prescaler value, which uses the system peripheral clock as the baseline and can be an even number in the range of 2–65535

- clock_mode: clock mode. Four modes are available based on different clock edge and phase configurations.
- rx_sample_delay: delayed clock cycle(s) of RX sampling. You can set it to “1” for the maximum frequency and “0” for other frequencies.

3.3.1.2 app_qspi_deinit

Table 3-5 app_qspi_deinit API

Function Prototype	uint16_t app_qspi_deinit(app_qspi_id_t id)
Function Description	Deinitialize the QSPI module.
Parameter	id: QSPI module ID
Return Value	Refer to APP_DRV_ERR_CODE in the source code.
Remarks	

3.3.1.3 app_qspi_get_handle

Table 3-6 app_qspi_get_handle API

Function Prototype	qspi_handle_t *app_qspi_get_handle(app_qspi_id_t id)
Function Description	Obtain the QSPI control handle based on corresponding module ID.
Parameter	id: QSPI module ID
Return Value	Pointer to the QSPI control handle
Remarks	

3.3.1.4 app_qspi_dma_init

Table 3-7 app_qspi_dma_init API

Function Prototype	uint16_t app_qspi_dma_init(app_qspi_params_t *p_params)
Function Description	Initialize QSPI DMA mode.
Parameter	p_params: pointer to the initialization parameter structure
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_`Folder\drivers\inc\app_drv_error.h` for details.
Remarks	

3.3.1.5 app_qspi_dma_deinit

Table 3-8 app_qspi_dma_deinit API

Function Prototype	uint16_t app_qspi_dma_deinit(app_qspi_id_t id)
Function Description	Deinitialize QSPI DMA mode.
Parameter	id: QSPI module ID

Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2 APIs in Register Mode

In register mode, these APIs serve as generic interfaces to access any QSPI peripheral.

- Based on application scenarios, they are categorized into two groups: APIs with command control and APIs without command control.
- Based on data transmission modes, they are categorized into three groups: app_xxx_transmit_sync, app_xxx_transmit_async, and app_xxx_dma_transmit_async, corresponding to data transmission in polling mode, in interrupt mode, and in DMA mode respectively.

3.3.2.1 app_qspi_command_sync

Table 3-9 app_qspi_command_sync API

Function Prototype	uint16_t app_qspi_command_sync(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint32_t timeout)
Function Description	Transmit commands synchronously in polling mode.
Parameter	• id: QSPI module ID • p_cmd: Specify the buffer that stores commands to be transmitted. • timeout: timeout period (ms)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.2 app_qspi_command_async

Table 3-10 app_qspi_command_async API

Function Prototype	uint16_t app_qspi_command_async(app_qspi_id_t id, app_qspi_command_t *p_cmd)
Function Description	Transmit commands asynchronously in interrupt mode.
Parameter	• id: QSPI module ID • p_cmd: Specify the buffer that stores commands to be transmitted.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.3 app_qspi_dma_command_async

Table 3-11 app_qspi_dma_command_async API

Function Prototype	uint16_t app_qspi_dma_command_async(app_qspi_id_t id, app_qspi_command_t *p_cmd)
Function Description	Transmit commands asynchronously in DMA mode.
Parameter	• id: QSPI module ID • p_cmd: Specify the buffer that stores commands to be transmitted.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.4 app_qspi_command_receive_sync

Table 3-12 app_qspi_command_receive_sync API

Function Prototype	uint16_t app_qspi_command_receive_sync(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data, uint32_t timeout)
Function Description	Read data synchronously in polling mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be read. • timeout: timeout period (ms)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.5 app_qspi_command_receive_async

Table 3-13 app_qspi_command_receive_async API

Function Prototype	uint16_t app_qspi_command_receive_async(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data)
Function Description	Read data asynchronously in interrupt mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be read.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.6 app_qspi_dma_command_receive_async

Table 3-14 app_qspi_dma_command_receive_async API

Function Prototype	uint16_t app_qspi_dma_command_receive_async(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data)
Function Description	Read data asynchronously in DMA mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be read.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.7 app_qspi_command_transmit_sync

Table 3-15 app_qspi_command_transmit_sync API

Function Prototype	uint16_t app_qspi_command_transmit_sync(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data, uint32_t timeout)
Function Description	Transmit data synchronously in polling mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be transmitted. • timeout: timeout period (ms)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.8 app_qspi_command_transmit_async

Table 3-16 app_qspi_command_transmit_async API

Function Prototype	uint16_t app_qspi_command_transmit_async(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data)
Function Description	Transmit data asynchronously in interrupt mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be transmitted.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.

Remarks	

3.3.2.9 app_qspi_dma_command_transmit_async

Table 3-17 app_qspi_dma_command_transmit_async API

Function Prototype	uint16_t app_qspi_dma_command_transmit_async(app_qspi_id_t id, app_qspi_command_t *p_cmd, uint8_t *p_data)
Function Description	Transmit data asynchronously in DMA mode, with a control command sent first.
Parameter	• id: QSPI module ID • p_cmd: control command • p_data: Specify the buffer that stores data to be transmitted.
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.10 app_qspi_transmit_sync_ex

Table 3-18 app_qspi_transmit_sync_ex API

Function Prototype	uint16_t app_qspi_transmit_sync_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length, uint32_t timeout)
Function Description	Transmit data synchronously in polling mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode) • data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer that stores data to be transmitted. • length: length of data to be transmitted (in bytes) • timeout: timeout period (ms)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.11 app_qspi_transmit_async_ex

Table 3-19 app_qspi_transmit_async_ex API

Function Prototype	uint16_t app_qspi_transmit_async_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length)

Function Description	Transmit data asynchronously in interrupt mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode) • data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer that stores data to be transmitted. • length: length of data to be transmitted (in bytes)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_`Folder\drivers\inc\app_drv_error.h` for details.
Remarks	

3.3.2.12 app_qspi_dma_transmit_async_ex

Table 3-20 app_qspi_dma_transmit_async_ex API

Function Prototype	uint16_t app_qspi_dma_transmit_async_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length)
Function Description	Transmit data asynchronously in DMA mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode) • data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer that stores data to be transmitted. • length: length of data to be transmitted (in bytes)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_`Folder\drivers\inc\app_drv_error.h` for details.
Remarks	

3.3.2.13 app_qspi_receive_sync_ex

Table 3-21 app_qspi_receive_sync_ex API

Function Prototype	uint16_t app_qspi_receive_sync_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length , uint32_t timeout, uint32_t timeout)
Function Description	Receive data synchronously in polling mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode)

	• data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer to receive data. • length: length of data to be received (in bytes) • timeout: timeout period (ms)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.14 app_qspi_receive_async_ex

Table 3-22 app_qspi_receive_async_ex API

Function Prototype	uint16_t app_qspi_receive_async_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length)
Function Description	Receive data asynchronously in interrupt mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode) • data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer to receive data. • length: length of data to be received (in bytes)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.2.15 app_qspi_dma_receive_async_ex

Table 3-23 app_qspi_dma_receive_async_ex API

Function Prototype	uint16_t app_qspi_dma_receive_async_ex(app_qspi_id_t id, uint32_t qspi_mode, uint32_t data_width, uint8_t *p_data, uint32_t length)
Function Description	Receive data asynchronously in DMA mode; the sequence mode and data width are configurable.
Parameter	• id: QSPI module ID • qspi_mode: sequence mode for transmitting data. Options: QSPI_DATA_MODE_SPI (standard SPI mode), QSPI_DATA_MODE_DUALSPI (Dual SPI mode), and QSPI_DATA_MODE_QUADSPI (Quad SPI mode) • data_width: data width. Options: QSPI_DATASIZE_08_BITS, QSPI_DATASIZE_16_BITS, and QSPI_DATASIZE_32_BITS • p_data: Specify the buffer to receive data.

	• length: length of data to be received (in bytes)
Return Value	APP_DRV_xxx: Refer to the macro definition in SDK_Folder\drivers\inc\app_drv_error.h for details.
Remarks	

3.3.3 APIs in Memory Mapped Mode

Apart from register mode, NOR Flash and PSRAM can also work in memory mapped mode by configuring the following APIs, making peripheral access simpler and more efficient.

3.3.3.1 app_qspi_config_memory_mappped

Table 3-24 app_qspi_config_memory_mappped API

Function Prototype	bool app_qspi_config_memory_mappped(app_qspi_id_t id, app_qspi_mmap_device_t dev);
Function Description	Set the device to memory mapped mode for transmitting data blocks.
Parameter	• id: QSPI module ID • dev: Specify the device type to work in memory mapped mode.
Return Value	true/false
Remarks	

app_qspi_mmap_device_t definition and members are detailed as follows.

```
typedef struct {
    app_qspi_device_e                       dev_type;
    app_qspi_psram_mmap_wr_cmd_e            psram_wr;
    union {
        app_qspi_flash_mmap_rd_cmd_e        flash_rd;
        app_qspi_psram_mmap_rd_cmd_e        psram_rd;
    } rd;
    void * set;
} app_qspi_mmap_device_t;
```

dev_type

- APP_QSPI_DEVICE_FLASH: NOR Flash
- APP_QSPI_DEVICE_PSRAM: PSRAM

psram_wr

Specify the write command for PSRAM when “dev_type” is configured as “APP_QSPI_DEVICE_PSRAM”. Options include

- PSRAM_MMAP_CMD_QWRITE_02H
- PSRAM_MMAP_CMD_QWRITE_38H

rd

Specify the read command for PSRAM/NOR Flash.

When NOR Flash is adopted, options for the read command flash_rd include

- FLASH_MMAP_CMD_DREAD_3BH: 3B command in Dual SPI mode
- FLASH_MMAP_CMD_2READ_BBH: BB command in Dual SPI mode
- FLASH_MMAP_CMD_QREAD_6BH: 6B command in Quad SPI mode
- FLASH_MMAP_CMD_4READ_EBH: EB command in Quad SPI mode

When PSRAM is adopted, options for the read command psram_rd include

- PSRAM_MMAP_CMD_QREAD_0BH: 0B command in Quad SPI mode
- PSRAM_MMAP_CMD_QREAD_EBH: EB command in Quad SPI mode

3.3.3.2 app_qspi_active_memory_mappped

Table 3-25 app_qspi_active_memory_mappped API

Function Prototype	bool app_qspi_active_memory_mappped(app_qspi_id_t id, bool is_active)
Function Description	Activate/Deactivate memory mapped mode.
Parameter	• id: QSPI module ID • is_active: Activate/Deactivate memory mapped mode.
Return Value	true/false
Remarks	

3.3.3.3 app_qspi_get_xip_base_address

Table 3-26 app_qspi_get_xip_base_address API

Function Prototype	uint32_t app_qspi_get_xip_base_address(app_qspi_id_t id)
Function Description	Obtain the base address in system bus mapped with QSPI memory.
Parameter	id: QSPI module ID
Return Value	Base address in system bus mapped with QSPI memory
Remarks	

3.3.3.4 app_qspi_mmap_set_endian_mode

Table 3-27 app_qspi_mmap_set_endian_mode API

Function Prototype	bool app_qspi_mmap_set_endian_mode(app_qspi_id_t id, app_qspi_mmap_endian_mode_e mode)
Function Description	Configure the data endian mode for read operations in memory mapped mode.
Parameter	• id: QSPI module ID • mode: data endian mode for read operations
Return Value	true/false

Remarks	

3.3.4 Special APIs

Special APIs are the ones combining key features of QSPI and DMA modules, making them an efficient choice for such scenarios as display refresh.

3.3.4.1 app_qspi_async_draw_screen

Table 3-28 app_qspi_async_draw_screen API

Function Prototype	bool app_qspi_async_draw_screen (app_qspi_id_t screen_id, app_qspi_id_t storage_id, const app_qspi_screen_command_t * const p_screen_cmd, const app_qspi_screen_info_t * const p_screen_info, app_qspi_screen_scroll_t * p_scroll_config, bool is_first_call)
Function Description	Asynchronous display drawing API
Parameter	• screen_id: display ID • storage_id: Specify ID of the QSPI memory that stores materials. • p_screen_cmd: pointer to the display refresh control command • p_screen_info: display information • p_link_scroll: display scrolling control information • is_first_call: To call this API in a foreground task, you shall set this parameter to “true”.
Return Value	true/false
Remarks	• storage_id shall be the same as “APP_STORAGE_RAM_ID” when graphic materials are not stored in the QSPI memory. • storage_id and screen_id cannot be the same. • scrn_pixel_depth of the parameter “p_screen_info” can be set to “2” only. That is, only 16-bit display output is supported by this API. • Display drive API is extracted in SDK_Folder\drivers\inc\app_graphics_qspi.c.

3.3.4.2 app_qspi_async_veri_draw_screen

Table 3-29 app_qspi_async_veri_draw_screen API

Function Prototype	bool app_qspi_async_veri_draw_screen(app_qspi_id_t screen_id, app_qspi_id_t storage_id, const app_qspi_screen_command_t * const p_screen_cmd, const app_qspi_screen_info_t * const p_screen_info, app_qspi_screen_veri_link_scroll_t * p_link_scroll, bool is_first_call)

Function Description	API specific for (vertical) chaining asynchronous display refresh
Parameter	• screen_id: display ID • storage_id: Specify ID of the QSPI memory that stores materials. • p_screen_cmd: pointer to the display refresh control command • p_screen_info: display information • p_link_scroll: pointer to the linked list structure of scrolling parameters • is_first_call: To call this API in a foreground task, you shall set this parameter to “true”.
Return Value	• true: succeeded • false: failed
Remarks	• storage_id shall be the same as “APP_STORAGE_RAM_ID” when graphic materials are not stored in the QSPI memory. • storage_id and screen_id cannot be the same. • scrn_pixel_depth of the parameter “p_screen_info” can be set to “2” only. That is, only 16-bit display output is supported by this API. • Display drive API is extracted in SDK_`Folder\drivers\inc\app_graphics_qspi.c`.

3.3.4.3 app_qspi_mmap_blit_image

Table 3-30 app_qspi_mmap_blit_image API

Function Prototype	bool app_qspi_mmap_blit_image(app_qspi_id_t storage_id, blit_image_config_t * p_blit_config, blit_xfer_type_e xfer_type)
Function Description	Transfer two-dimensional data through DMA.
Parameter	• storage_id: Specify ID of the QSPI memory that stores materials. • p_blit_config: blit configuration • xfer_type: Specify the transfer type. Options: SG and LLP
Return Value	• true: succeeded • false: failed
Remarks	• Display drive API is extracted in SDK_`Folder\drivers\inc\app_graphics_qspi.c`. • The macro QSPI_BLIT_RECT_IMAGE_SUPPORT in *app_qspi_user_config.h* shall be set to a value larger than 0 (default: 0) before this API is called.

3.3.4.4 app_qspi_async_llp_draw_block

Table 3-31 app_qspi_async_llp_draw_block API

Function Prototype	bool app_qspi_async_llp_draw_block(app_qspi_id_t screen_id, app_qspi_id_t storage_id, const app_qspi_screen_command_t *const p_screen_cmd,

	const app_qspi_screen_info_t *const p_screen_info, app_qspi_screen_block_t *p_block_info, bool is_first_call)
Function Description	Refresh the display block areas by means of DMA LLP (DMA chain). Each node of the linked list corresponds to a line of data in the graphic data block(s) to be refreshed.
Parameter	• screen_id: display ID • storage_id: Specify ID of the QSPI memory that stores materials. • p_screen_cmd: pointer to the display refresh control command • p_screen_info: display information • p_block_info: graphic data block(s) to be refreshed • is_first_call: When this API is called for the first time, you shall set it to “true”, which means a command shall be transmitted before a frame of graphic materials.
Return Value	• true: succeeded • false: failed
Remarks	When data transmission completes, the APP_QSPI_EVT_TX_CPLT event is triggered.

3.4 QSPI Operational Recommendations

1. System peripheral clock that is the QSPI clock source serves as the baseline of QSPI clock prescaler value. The QSPI clock prescaler value shall be an even number in the range of 2–65535.
2. For data transmission of SRAM or QSPI devices in memory mapped mode, when the data to be transmitted is less than 1 KB, the memcpy function is recommended in most cases; DMA transmission is recommended when the data size is larger than 1 KB. DMA has obvious advantages on transmitting a large amount of data.
3. Before the memcpy/memset function is adopted to access data in memory mapped mode, you are recommended to compile system standard library instead of MicroLib in Keil for the sake of more efficient access, because the former can better activate the bus burst transfer.
4. Some QSPI sequences require enough Tcsu (CS Setup Delay Time) to ensure stable data access. You can
   - Set the clock mode to Clock Mode 3 if supported. Generally, Clock Mode 3 has longer Tcsu than Clock Mode 0.
   - Increase Tcsu by configuring the specific register.
5. It is necessary for QSPI PSRAM to release CS regularly to perform self-refresh of the internal data retention circuit. The longest CS time is defined as “tCEM”, which shall be complied with strictly, especially for write operations.
   - For write operations in memory mapped mode, tCEM is taken into account in design, and no configuration is required.
   - For write operations in register mode, tCEM is managed by the driver layer. QSPI PSRAM is not recommended in this mode due to the complex API design.

6. QSPI supports data pre-fetch mode, in which optimal read efficiency of QSPI can be ensured in read operations. Note that this mode can only work with DMA, and cannot be enabled during CPU access.
7. In GR5526 SoC design, differentiated optimization to each QSPI/DMA FIFO is performed, making it feasible to connect peripherals randomly in most cases. To highlight the optimal efficiency of GR5526 series, recommended connection schemes are as follows:
   - QSPI0: Connect Flash preferentially; work with channel 0 of DAM0.
   - QSPI1: Connect PSRAM preferentially; work with channel 0 of DAM0/DAM1.
   - QSPI2: Connect a display device preferentially; work with channel 0 of DAM1.
   - It is recommended to allocate channel 0 of DAM0/DAM1 to peripherals with high throughput such as QSPI0/QSPI1/QSPI2, and not to adopt the two channels simultaneously.

4 Typical Display Controller Features

4.1 Introduction

Mainly used to work with GPU, Display Controller transmits the rendered frame buffer to the display for graphics display. Typical color formats supported by the frame buffer include RGBA8888, RGB565, and TSCx, and the display interface specification is MIPI DBI Type-C, which covers the following sequence protocols:

- 3-wire SPI

Include three signal lines: SPI_CS (chip enable), SPI_SCLK (serial clock), and SPI_SD (serial data output). The command word consists of 9 bits: 1-bit data/command indicator and 8-bit command. The data word consists of 1-bit data/command indicator and several bits of pixel data.

- 4-wire SPI (extended DCX signal line)

Include four signal lines: SPI_CS (chip enable), SPI_SCLK (serial clock), SPI_SD (serial data output), and SPI_DC (data/command indicator). The command word consists of 8 bits. The data word consists of several bits of pixel data.

- Dual SPI

Include four signal lines: SPI_CS (chip enable), SPI_SCLK (serial clock), and SPI_SD & SPI_SD1 (serial data outputs). The command word is always transmitted through the SPI_SD line, and the data is routed through the SPI_SD and SPI_SD1 lines. The command word consists of 9 bits: 1-bit data/command indicator and 8-bit command. The data word consists of 1-bit data/command indicator and several bits of pixel data.

- Quad SPI

Include six signal lines: SPI_CS (chip enable), SPI_SCLK (serial clock), as well as SPI_SD, SPI_SD1, SPI_SD2, and SPI_SD3 (serial data outputs). The command type is always transmitted through the SPI_SD line or through the SPI_SDx lines. QSPI sequence generally consists of 8-bit command header, 24-bit command word, and several bits of pixel data.

The Display Controller is incorporated with DMA module, to speed up the transmission of frame data pairs to display. DC is also capable of transmitting an entire frame data pair at one time.

Note:

Display Controller is often abbreviated as DC in this document. Special attention shall be paid to distinguish it with Direct Current, which can also be abbreviated as DC.

4.2 Input Color Format

DC provides the input end with rich sets of color formats. Color formats applied to wearable products are as follows:

1. RGBA 8888 32-bit

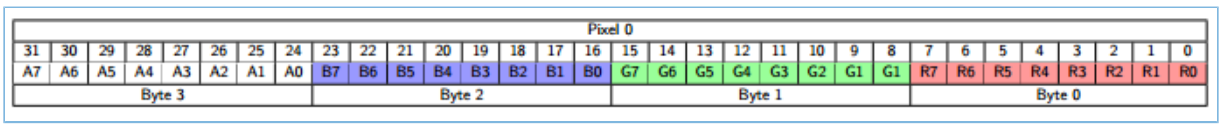

Figure 4-1 RGBA 8888 32-bit format

2. ARGB 8888 32-bit

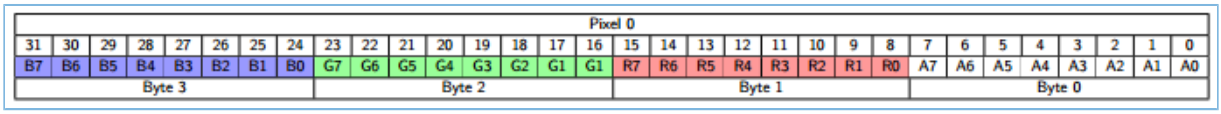

Figure 4-2 ARGB 8888 32-bit

3. ABGR 8888 32-bit

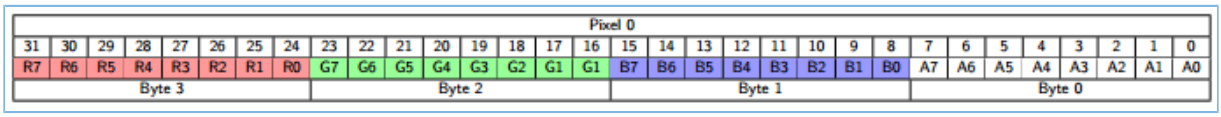

Figure 4-3 ABGR 8888 32-bit

4. BGRA 8888 32-bit

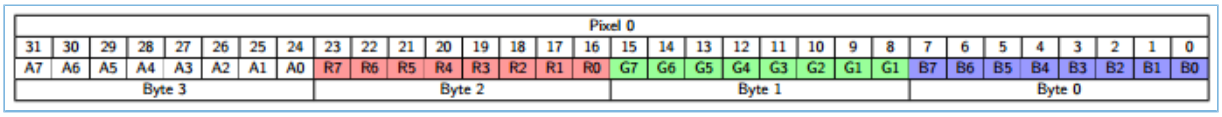

Figure 4-4 BGRA 8888 32-bit

5. RGBA 5551 16-bit

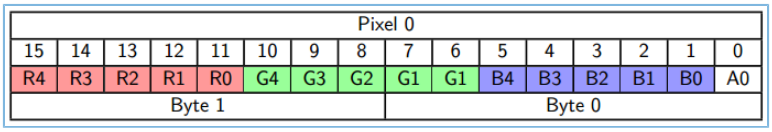

Figure 4-5 RGBA 5551 16-bit format

6. RGB 565 16-bit

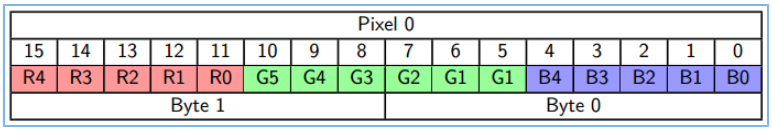

Figure 4-6 RGB 565 16-bit format

7. TSCx compression format

   The TSC compression is a fixed-length lossy compression format in units of 4 x 4 pixel blocks, covering TSC4, TSC6, and TSC6A.

   - TSC4: Pixel after compression is 4 bpp, represented as 4 bits.
   - TSC6/TSC6A: Pixel after compression is 6 bpp, represented as 6 bits. TSC6A carries alpha information, whereas TSC6 does not.

4.3 Display Color Format

Display color format carries no alpha information. For MIPI DBI Type-C display interface of wearable products, the display format sequence in frequent use is composed of three parts, XXX – YYY – ZZZ, as detailed below:

- XXX: Represent SPI protocol type. For example, 3-wire, 4-wire, Dual SPI, and Quad SPI.
- YYY: Represent color format.
- ZZZ: Represent sequence option. Transmission signal sequence may be different with the same XXX and YYY. Thus, this parameter is required for distinguishing the sequence.

1. 3-wire SPI – RGB565 – Option0

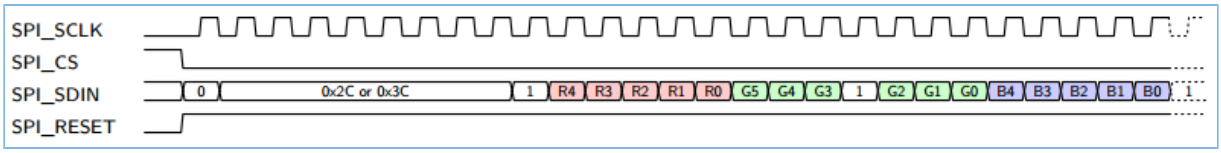

Figure 4-7 3-wire SPI – RGB565 – Option0 format

2. 4-wire SPI – RGB565 – Option0

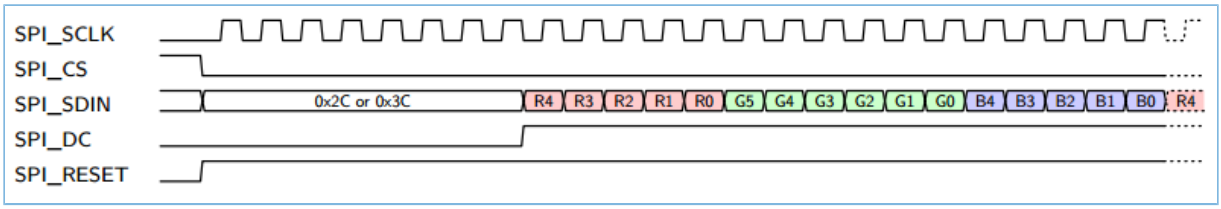

Figure 4-8 4-wire SPI – RGB565 – Option0 format

3. 3-wire SPI – RGB888 – Option0

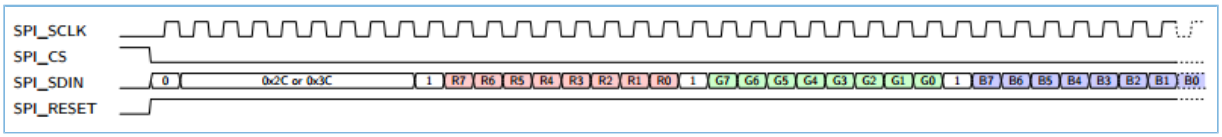

Figure 4-9 3-wire SPI – RGB888 – Option0 format

4. 4-wire SPI – RGB888 – Option0

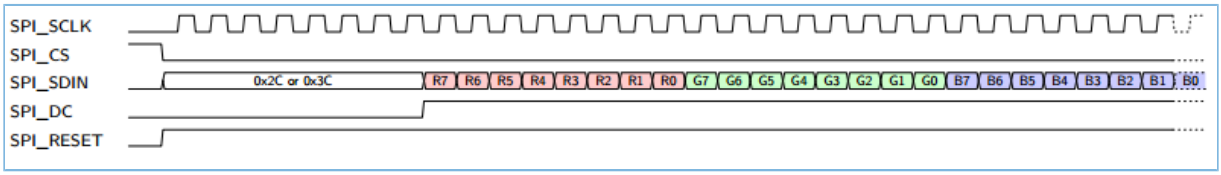

Figure 4-10 4-wire SPI – RGB888 – Option0 format

5. Dual SPI – RGB565 – Option0

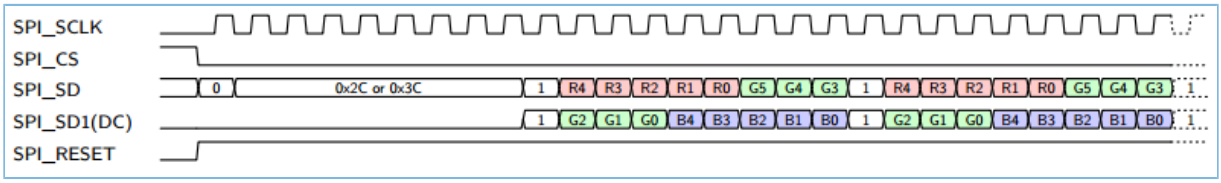

Figure 4-11 Dual SPI – RGB565 – Option0 format

6. Dual SPI – RGB888 – Option0

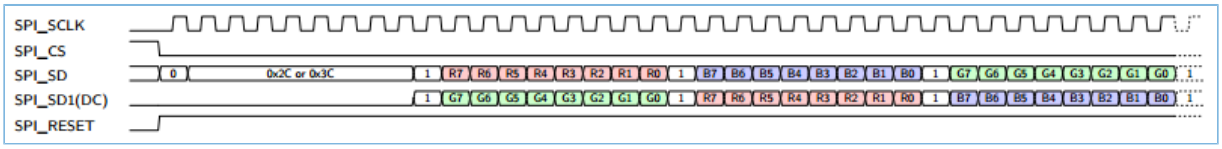

Figure 4-12 Dual SPI – RGB888 – Option0 format

7. Dual SPI – RGB888 – Option1

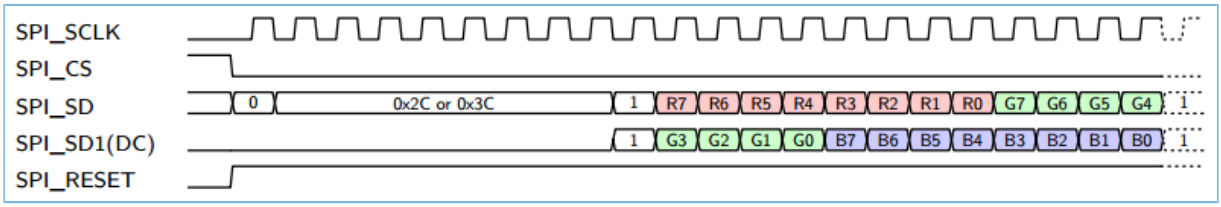

Figure 4-13 Dual SPI – RGB888 – Option1 format

8. Quad SPI – RGB565 – Option0

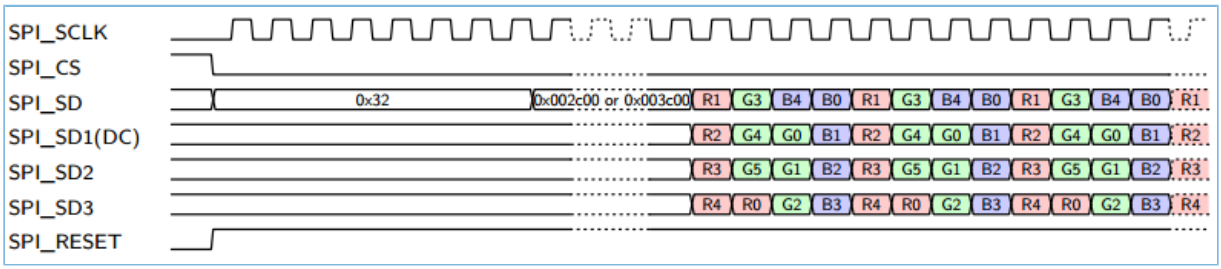

Figure 4-14 Quad SPI – RGB565 – Option0 format

9. Quad SPI – RGB888 – Option0

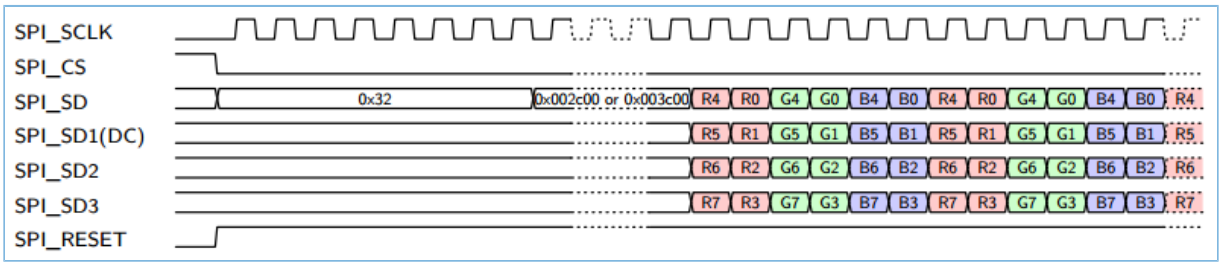

Figure 4-15 Quad SPI – RGB888 – Option0 format

4.4 Basic Operating Logic

Basic operating logic of display refresh operation is shown below.

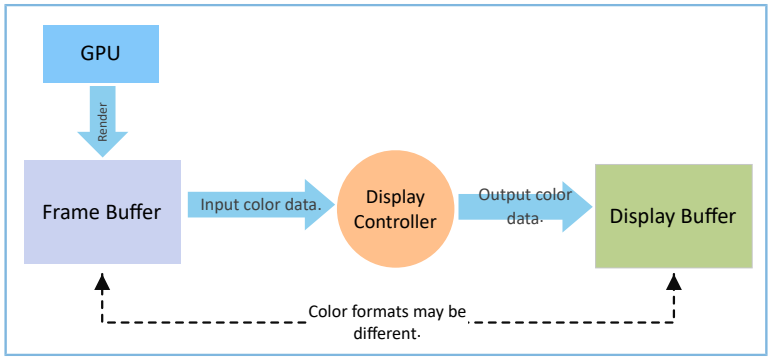

Figure 4-16 Basic operating logic

1. Display Controller specifies the input frame buffer, including display buffer address, pixel width/height, and color depth/format.

   (1). The frame buffer can be either the one rendered by GPU or source graphics with specific format(s).

   (2). The maximum theoretical pixel of the frame buffer is 800 x 600, and the practical pixel is determined by the resolution of wearable products in use.

   (3). If the frame buffer data is in uncompressed format, the frame buffer address shall be 4-byte aligned; for a TSCx compression format, the frame buffer address shall be 16-byte aligned, and the pixel width as well as the height shall be an integral multiple of 4.

2. Display Controller shall also specify the output specifications such as color format and sequence format.

4.5 Driver API Introduction

4.5.1 graphics_dc_init

Table 4-1 graphics_dc_init API

Function Prototype	uint16_t graphics_dc_init(app_graphics_dc_params_t * dc_params, graphics_dc_irq_event_notify_cb evt_cb);
Function Description	Initialize the DC module.
Parameter	Refer to the parameter description below.
Return Value	N/A
Remarks	

Parameter description:

app_graphics_dc_params_t * dc_params: Configure Display Controller initialization parameters.

1. app_graphics_dc_params_t members are detailed below:

   - graphics_dc_mspi_e mspi_mode: SPI sequence mode of the display
     - GDC_MODE_SPI: SPI mode
     - GDC_MODE_DSPI: 4-wire SPI mode

- GDC_MODE_QSPI: Quad SPI mode

- graphics_dc_clock_freq_e clock_freq: Configure the DC clock frequency.
  - GDC_CLOCK_FREQ_48MHz: DC clock, 48 MHz
  - GDC_CLOCK_FREQ_24MHz: DC clock, 24 MHz
  - GDC_CLOCK_FREQ_12MHz: DC clock, 12 MHz
  - GDC_CLOCK_FREQ_6MHz: DC clock, 6 MHz
  - GDC_CLOCK_FREQ_3MHz: DC clock, 3 MHz
- graphics_dc_clock_mode_e clock_mode: sequence clock mode
  - GDC_CLOCK_MODE_0: clock mode 0
  - GDC_CLOCK_MODE_1 clock mode 1
  - GDC_CLOCK_MODE_2: clock mode 2
  - GDC_CLOCK_MODE_3: clock mode 3
- graphics_dc_tcsu_cycle_e tcsu_cycle: the number of the delayed clock cycle(s) before CS establishment
  - GDC_TCSU_CYCLE_0: Delay 0 clock cycle.
  - GDC_TCSU_CYCLE_1: Delay 1 clock cycle.
  - GDC_TCSU_CYCLE_2: Delay 2 clock cycles.
  - GDC_TCSU_CYCLE_3: Delay 3 clock cycles.
  - GDC_TCSU_CYCLE_4: Delay 4 clock cycles.
- graphics_dc_layer_mode_e layer_mode: layer mode. The currently available configuration is “GDC_ONE_LAYER_MODE”.
- graphics_dc_mipi_format_e mipicfg_format: frame output sequence/protocol format
  - GDC_MIPICFG_SPI_RGB565_OPT0: Use SPI to transmit frame, and output the frame in RGB565 – option0 format.
  - GDC_MIPICFG_SPI_RGB888_OPT0: Use SPI to transmit frame, and output the frame in RGB888 – option0 format.
  - GDC_MIPICFG_DSPI_RGB565_OPT0: Use DSPI to transmit frame, and output the frame in RGB565 – option0 format.
  - GDC_MIPICFG_DSPI_RGB888_OPT0: Use DSPI to transmit frame, and output the frame in RGB888 – option0 format.
  - GDC_MIPICFG_DSPI_RGB888_OPT1: Use DSPI to transmit frame, and output the frame in RGB888 – option1 format.

- GDC_MIPICFG_QSPI_RGB565_OPT0: Use QSPI to transmit frame, and output the frame in RGB565 – option0 format.
- GDC_MIPICFG_QSPI_RGB888_OPT0: Use QSPI to transmit frame, and output the frame in RGB888 – option0 format.

- uint16_t resolution_x: frame buffer resolution on the X-axis
- uint16_t resolution_y: frame buffer resolution on the Y-axis
- app_graphics_dc_pins_t pins_cfg: DC I/O configuration

2. graphics_dc_irq_event_notify_cb evt_cb

   It is a user-defined callback that needs to be registered during system initialization. The callback is executed asynchronously in display refresh applications by Display Controller.

4.5.2 app_graphics_dc_spi_send

Table 4-2 app_graphics_dc_spi_send API

Function Prototype	void app_graphics_dc_spi_send(uint8_t cmd_8bit, uint32_t address_24bit, uint8_t * data, uint32_t length);
Function Description	Transmit control data in 3-wire SPI mode. It is mainly used for initialization configuration of the display.
Parameter	• cmd_8bit: 8-bit command header • address_24-bit: 24-bit command word/command control address • data: data to be transmitted • length: length of data to be transmitted
Return Value	None
Remarks	The length of data to be transmitted shall not exceed 8 bytes at a time.

Sequence example:

```
unit8_t data[4] = {0x00, 0x7E,0x01,0x45}
app_graphics_dc_spi_send (0x02, 0x022A00,&data[0],4)
```

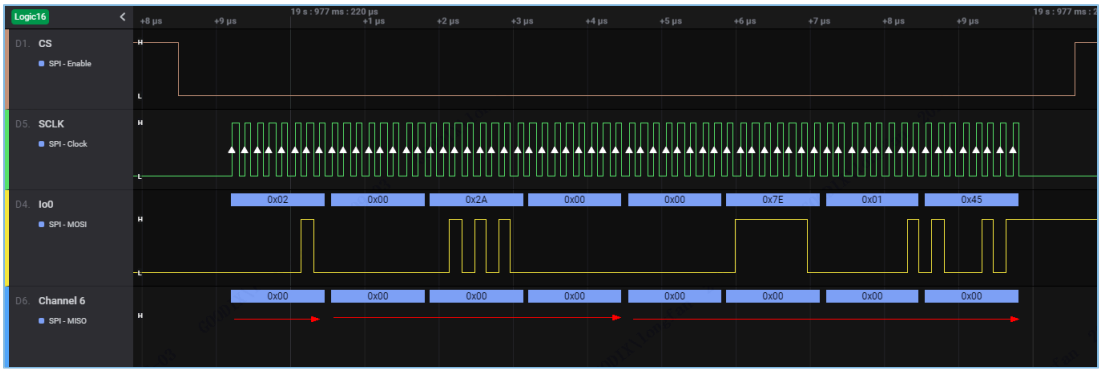

Figure 4-17 app_graphics_dc_spi_send sequence

4.5.3 app_graphics_dc_dspi_send_cmd_in_3wire_1lane

Table 4-3 app_graphics_dc_dspi_send_cmd_in_3wire_1lane API

Function Prototype	void app_graphics_dc_dspi_send_cmd_in_3wire_1lane(uint8_t cmd);
Function Description	Transmit 9-bit command word in 3-wire SPI mode. It is mainly used for initialization configuration of the display.
Parameter	cmd: 8-bit command word to be transmitted (the ninth bit is supplemented by the Display Controller)
Return Value	None
Remarks	

Sequence example:

```
app_graphics_dc_dspi_send_cmd_in_3wire_1lane(0x11);
```

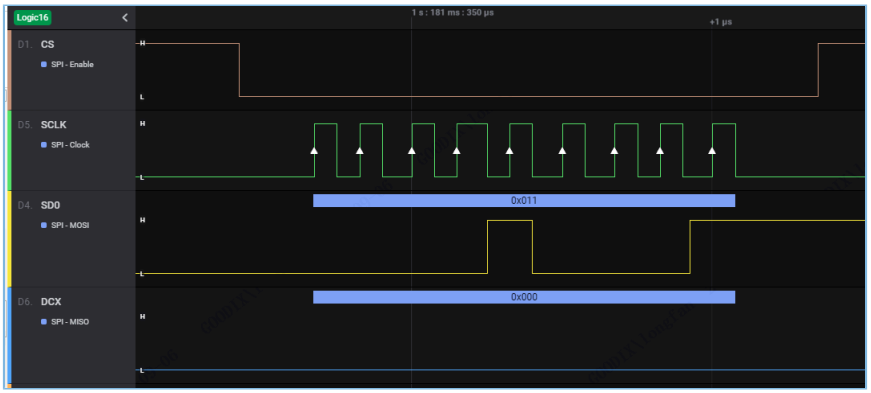

Figure 4-18 app_graphics_dc_dspi_send_cmd_in_3wire_1lane sequence

4.5.4 app_graphics_dc_dspi_send_cmd_data_in_3wire_1lane

Table 4-4 app_graphics_dc_dspi_send_cmd_data_in_3wire_1lane API

Function Prototype	void app_graphics_dc_dspi_send_cmd_data_in_3wire_1lane(uint8_t cmd, uint8_t data) ;
Function Description	Transmit a 9-bit command word and a piece of data in 3-wire SPI mode. It is mainly used for initialization configuration of the display.
Parameter	• cmd: command word to be transmitted (the ninth bit is supplemented by the Display Controller.) • data: data to be transmitted (the ninth bit is supplemented by the Display Controller.)
Return Value	None
Remarks	

Sequence example:

```
app_graphics_dc_dspi_send_cmd_data_in_3wire_1lane(0xE7, 0x10);
```

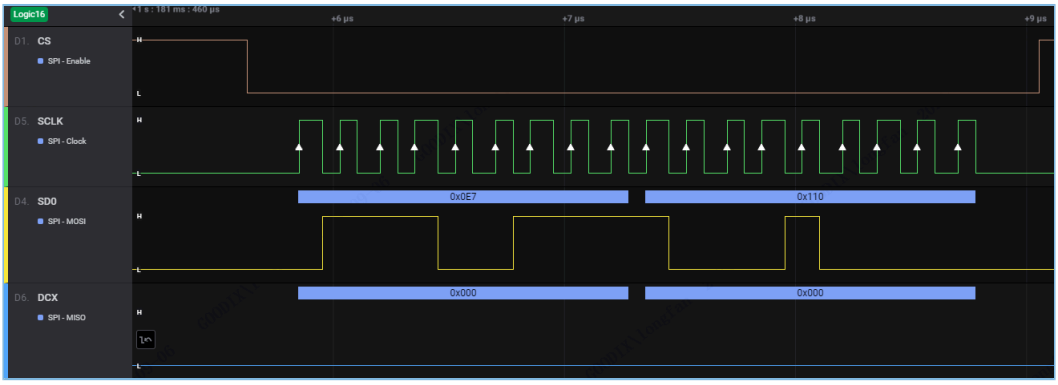

Figure 4-19 app_graphics_dc_dspi_send_cmd_in_3wire_1lane sequence

4.5.5 app_graphics_dc_dspi_send_cmd_data_in_4wire_2lane

Table 4-5 app_graphics_dc_dspi_send_cmd_data_in_4wire_2lane API

Function Prototype	void app_graphics_dc_dspi_send_cmd_data_in_4wire_2lane(uint16_t cmd, uint16_t data);
Function Description	Transmit a command word and a piece of data in 4-wire SPI mode. It is mainly used for initialization configuration of the display.
Parameter	• cmd: command word to be transmitted. The lower 8 bits are filled with “0” to extend to 16 bits. • data: data to be transmitted. The lower 8 bits are filled with “0” to extend to 16 bits.
Return Value	None
Remarks	

Sequence example:

```
// Write Command 0x36 and Data 0x00
app_graphics_dc_dspi_send_cmd_data_in_4wire_2lane(0x3600, 0x0000);
```

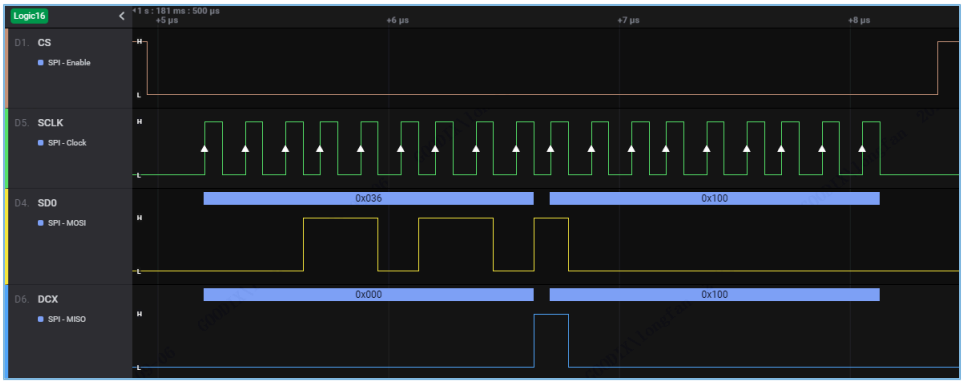

Figure 4-20 app_graphics_dc_dspi_send_cmd_data_in_4wire_2lane sequence

4.5.6 app_graphics_dc_dspi_send_cmd_datas_in_4wire_2lane

Table 4-6 app_graphics_dc_dspi_send_cmd_datas_in_4wire_2lane API

Function Prototype	void app_graphics_dc_dspi_send_cmd_datas_in_4wire_2lane(uint16_t cmd, uint16_t * data , int length);

Function Description	Transmit a command word and several pieces of data in 4-wire SPI mode. It is mainly used for initialization configuration of the display.
Parameter	• cmd: command word to be transmitted. The lower 8 bits are filled with “0” to extend to 16 bits. • data: data pointer. The lower 8 bits of all data are filled with “0” to extend to 16 bits. • length: the number of data to be transmitted
Return Value	None
Remarks	The number of data to be transmitted shall not exceed 8 at a time.

Sequence example:

```
// Write to 1 Command 0xB2 and 5 hex Data [0C,0C,00,33,33]
uint16_t b2_data[5] = {0x0C00, 0x0C00, 0x0000, 0x3300, 0x3300}; // extend to 16 bits
app_graphics_dc_dspi_send_cmd_datas_in_4wire_2lane(0xB200, b2_data, 5);
```

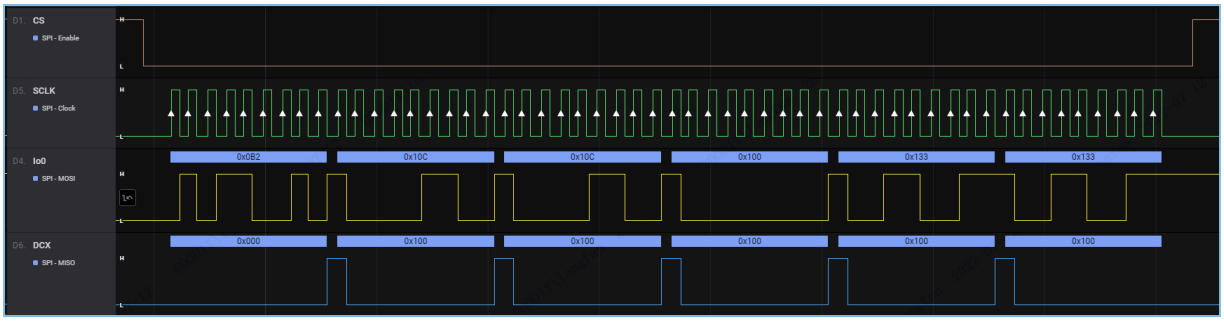

Figure 4-21 app_graphics_dc_dspi_send_cmd_datas_in_4wire_2lane sequence

4.5.7 app_graphics_dc_send_single_frame

Table 4-7 app_graphics_dc_send_single_frame API

Function Prototype	app_graphics_dc_frame_result_e app_graphics_dc_send_single_frame(uint32_t which_layer, app_graphics_dc_framelayer_t * frame_layer, app_graphics_dc_cmd_t * dc_cmd, app_graphics_dc_access_type_e access_type);
Function Description	Transmit single-frame data.
Parameter	Refer to the parameter description below.
Return Value	None
Remarks	

Parameter description:

- which_layer:
  - GRAPHICS_DC_LAYER_0: Display Controller layer 0
  - GRAPHICS_DC_LAYER_1: Display Controller layer 1

  Currently, only GRAPHICS_DC_LAYER_0 is supported.

- app_graphics_dc_framelayer_t * frame_layer: Configure the frame buffer.

  app_graphics_dc_framelayer_t members are detailed below:

  - frame_baseaddr: pointer to the frame buffer address. Note that 4-byte alignment is required for non-compressed format; 16-byte alignment is required for TSC compression format.
  - resolution_x: frame buffer resolution (pixel width) on the X-axis
  - resolution_y: frame buffer resolution (pixel height) on the Y-axis
  - row_stride: row stride, calculated by the formula: Resolution on the X-axis x Byte(s) per pixel. Setting to "-1" indicates it is calculated by the driver layer.
  - start_x: the origin on the X-axis. Generally, it is set to 0.
  - start_y: the origin on the Y-axis. Generally, it is set to 0.
  - size_x: X-axis drawing size, which is generally the same as resolution_x.
  - size_y: Y-axis drawing size, which is generally the same as resolution_y.
  - alpha: Blend global alpha. It is reserved and shall be set to 0xff.
  - blendmode: blend mode. Currently, it shall be set to HAL_GDC_BL_SRC.
  - data_format: pixel color format of the frame buffer. The following formats are currently supported.
    - GDC_DATA_FORMAT_RGB565: RGB565 format
    - GDC_DATA_FORMAT_RGBA8888: RGBA8888
    - GDC_DATA_FORMAT_ABGR8888: ABGR8888
    - GDC_DATA_FORMAT_ARGB8888: ARGB8888
    - GDC_DATA_FORMAT_BGRA8888: BGRA8888
    - GDC_DATA_FORMAT_TSC4: TSC4 compression format
    - GDC_DATA_FORMAT_TSC6: TSC6 compression format
    - GDC_DATA_FORMAT_TSC6A: TSC6A compression format
- app_graphics_dc_cmd_t * dc_cmd: DC output control command

  app_graphics_dc_cmd_t members are detailed below:

  - command: display control command
  - address: display control address
  - address_width: display control address width
    - GDC_FRAME_ADDRESS_WIDTH_NONE: None
    - GDC_FRAME_ADDRESS_WIDTH_08BIT: 8 bits
    - GDC_FRAME_ADDRESS_WIDTH_16BIT: 16 bits

        - GDC_FRAME_ADDRESS_WIDTH_24BIT: 24 bits
    - frame_timing: Select the frame output sequence.
        - GDC_SPI_FRAME_TIMING_0: Output through SPI sequence, which consists of 8-bit command, 24-bit address, and several bits of pixel data.
        - GDC_DSPI_FRAME_TIMING_0: Output through 4-wire SPI sequence (with DCX signal), which consists of 8-bit command, no control address, and several bits of pixel data.
        - GDC_QSPI_FRAME_TIMING_0: Output through Quad SPI sequence. 8-bit command, 24-bit control address, and several bits of pixel data are transmitted by QSPI.
        - GDC_QSPI_FRAME_TIMING_1: Output through Quad SPI sequence. 8-bit command is transmitted by SPI; 24-bit control address and several bits of pixel data are transmitted by QSPI.
- app_graphics_dc_access_type_e access_type: Specify the API access type.
    - GDC_ACCESS_TYPE_SYNC: synchronous mode. The function cannot return until data transmission completes. System CPU utilization efficiency is reduced in this mode; thus, it is generally for debugging only.
    - GDC_ACCESS_TYPE_ASYNC: asynchronous mode. In this mode, users need to pay attention to the function implementation result from the callback API.