RTL8762E Flash User Guide

V1.0

2021/11/16

Revision History

Date	Version	Comments	Author	Reviewer
2021/11/16	V1.0	First release version	Yueming	Serval

Contents

Figure List

1 Introduction

Flash is a non-volatile storage. Compare to RAM, which could be written or read directly, flash must be empty or erased before being written. With programming time limit up to 100K, flash can develop bad blocks when same blocks been erased frequently. It is recommended to use FTL (flash translation layer) rather than flash driver for user data storage. If really needed, APIs with "_locked" suffix are preferred.

RTL8762E supports executing code (XIP) on SPI Flash, but there are also some limitations and precautions.

2 Basic Operations

There are three basic operations for flash – read, write, and erase. All operation interfaces suffixed with "_locked" mean that all flash operations must take the semaphore before accessing flash, and release it after completion. That is because there is only one resource (flash) and only one path (flash controller) to access it. This helps prevent multiple tasks or interrupts from destroying the atomicity of the SPIC command sequence.

2.1 Flash API

2.1.1 Read

- **bool flash_auto_read_locked(uint32_t addr, uint32_t *data);**

flash_auto_read_locked reads out 4 bytes from flash. This API is faster than flash_read_locked, so it is recommended to use flash_auto_read_locked rather than flash_read_locked when possible. Sequential transfer is available for faster read speed with flash_ioctl api.

- **bool flash_read_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);**

flash_read_locked reads data from flash through user mode. Users could specify data length to be read out.

2.1.2 Write

- **bool flash_auto_write_locked(uint32_t start_addr, uint32_t data);**

flash_auto_write_locked writes 4 bytes to flash through auto mode.

- **bool flash_auto_write_buffer_locked(uint32_t start_addr, uint32_t *data, uint32_t len);**

flash_auto_write_buffer_locked supports write bunch of data to flash through auto mode. Input data and start address must be 4 bytes align.

- **bool flash_write_locked(uint32_t start_addr, uint32_t data_len, uint8_t *data);**

flash_write_locked writes data to flash through user mode. Users could specify data length to be written. There is no start address and data alignment limitations.

2.1.3 Erase

- **bool flash_erase_locked(T_ERASE_TYPE type, uint32_t addr);**

flash_erase_locked supports 2 erase types (T_ERASE_TYPE), FLASH_ERASE_SECTOR (4K bytes) and FLASH_ERASE_BLOCKE (64K bytes). The embedded SPI Flash in RTL8762E supports chip erase (FLASH_ERASE_CHIP). There is no usage scenario of chip erase in a normal application, so chip erase is not supported.

2.2 Access Mode

Flash controller supplies two modes for these three basic operations – user mode and auto mode. To operate with flash, user needs to set registers and transmit serial data to flash. These series sequence steps are called user mode. For users to control easily, controller supports auto mode to access flash as accessing memory. In RTL8762E SDK, both interfaces of two access modes are provided. Interface of auto mode has the word "auto", and others indicate accessing flash in user mode. For example, using flash_auto_read_locked will read flash in auto mode, while read flash in user mode with flash_read_locked.

To speed up the flash accessing efficiency, RTL8762E supports 8KB cache. Flash controller supports two sets of mapped address spaces for at most 8MB size, corresponding to cache address space [0x800000, 0x1000000) and non-cache address space [0x1800000, 0x2000000). When cache is enabled (the macro "SHARE_CACHE_RAM_SIZE" for mem_config.h is configured to be 0K or 4K), user needs to pay attention to the following two points during auto mode read or write:

1. It is forbidden to write access to flash's cache address space in automatic mode, so when calling interfaces flash_auto_write_locked and flash_auto_write_buffer_locked, no matter the incoming address is cache address or non-cache address, it will be processed to non-cache address in flash driver.
2. Read flash in automatic mode. If you visit cache address, you can only read read-only data, such as code or const data. Once the RW type data is read, it is possible to read the old value in cache which is not flushed yet.

Auto mode is easy to use but with risk. Flash address starts from 0x1800000, and within its length (depends on selected flash model) can easily be accessed via auto mode. However, if programmer misuses the range and accesses RAM, flash data would be destroyed. That is why using BP to protect partial blocks for code sections and important data is needed. The locked space could not be written or erased, thus protecting the code area and important data area.

3 Bit Modes

Apart from the standard Serial Peripheral Interface (SPI), most flash models also support high performance Dual/Quad modes I/O SPI controlled by six pins :

- Serial Clock (CLK)
- Chip Select (CS#)
- Serial Data I/O0 (DI)
- Serial Data I/O1 (DO)
- Serial Data IO2 (WP#)
- Serial Data I/O3 (HOLD#)

3.1 Three Bit Modes

1. **Single Mode** – Standard SPI mode as called 1-bit mode, which only uses CLK, CS#, DI, and DO. WP# is still available for Write Protect Input, and HOLD# is also available for Hold Input.
2. **Dual Mode** – as called 2-bit mode, uses CLK, CS#, and also uses DI as IO0, DO as IO1. Same with Single mode, WP# and HOLD# are also available.
3. **Quad mode** – as called 4-bit mode, needs all six pins, CLK, CS#, DI as IO0, DO as IO1, WP# as IO2, and HOLD# as IO3. Since all pins are used, Write Protect and Hold functions do not work in Quad mode.

Although almost all flash models support Dual and Quad modes, the command set and mode switch rules are not all same.

3.2 Bit Mode Switch

In order to support as many flash models as possible, single mode (1-bit mode) is used at boot time. If users need to switch to high speed bit mode (2-bit or 4-bit mode), interface flash_try_high_speed is provided in the SDK to switch to high bit mode. The parameter "bit_mode" is used to configure bit mode, and return value of the function indicates whether the switch is successful.

If Dual mode (2-bit mode) or Quad mode (4-bit mode) is selected, flash is configured and calibration is performed. When calibration fails, bit mode will be switched back to Single mode (1-bit mode). It should be noted that additional pins P1_3 and P1_4 will be used as IO2 and IO3, and hardware circuits should also support when switching to 4-bit mode.

The prototype of the interface function provided by SDK for bit mode switching is as follows:

- **uint32_t flash_try_high_speed(T_FLASH_MODE bit_mode);**

This api returns bit mode switch information. When it returns 1, bit mode switch is success; 0 means current flash is not supported yet,; -7 means calibration failed.

4 Software Block Protect

Although flash supports HW protect pin (WP#) to lock all flash to prevent writing and erasing operations, there are still two disadvantages.

1. If WP# is used for flash protection, Quad mode (4-bit mode) is not allowed.
2. HW protection can just choose to protect all or protect none, can't protect partially.

A mechanism to solve these problems is flash software Blocks Protection (BP). It uses some BP bits in flash status register to select the level (range) to protect as described below.

Flash uses BP(x) bits in status registers to identify numbers of blocks to lock, and TB bit to determine the direction to lock. RTL8762E only supports locking from lower address of flash.

STATUS REGISTER(1)					W25Q16DV (16M-BIT) MEMORY PROTECTION(3)			
SEC	TB	BP2	BP1	BP0	PROTECTED BLOCK(S)	PROTECTED ADDRESSES	PROTECTED DENSITY	PROTECTED PORTION(2)
X	X	0	0	0	NONE	NONE	NONE	NONE
0	0	0	0	1	31	1F0000h – 1FFFFFh	64KB	Upper 1/32
0	0	0	1	0	30 and 31	1E0000h – 1FFFFFh	128KB	Upper 1/16
0	0	0	1	1	28 thru 31	1C0000h – 1FFFFFh	256KB	Upper 1/8
0	0	1	0	0	24 thru 31	180000h – 1FFFFFh	512KB	Upper 1/4
0	0	1	0	1	16 thru 31	100000h – 1FFFFFh	1MB	Upper 1/2
0	1	0	0	1	0	000000h – 00FFFFh	64KB	Lower 1/32
0	1	0	1	0	0 and 1	000000h – 01FFFFh	128KB	Lower 1/16
0	1	0	1	1	0 thru 3	000000h – 03FFFFh	256KB	Lower 1/8
0	1	1	0	0	0 thru 7	000000h – 07FFFFh	512KB	Lower 1/4
0	1	1	0	1	0 thru 15	000000h – 0FFFFFh	1MB	Lower 1/2
X	X	1	1	X	0 thru 31	000000h – 1FFFFFh	2MB	ALL

Figure 4-1 Flash block protect

Here we just use BP to protect some important data such as configuration, security, and code sections, not all

portions will be protected. BP function is not available for customers because different flash vendors and models have different rules and limitation. Accessing status registers frequently will damage flash. RTL8762E provides a configuration option "bp_enable" to determine whether or not to enable the BP function (disabled by default).

Flash status register stores data as NVRAM type by default. BP function needs to change (write) BP bits in order to switch to different protect level, but NVRAM has 100K times programming limitation. Although most vendors support 0x50 command to switch flash to SRAM type, but some vendors such as MXIC doesn't support it..

At present, if BP function is enabled by configuring macro FLASH_BLOCK_PROTECT_ENABLE in otp_config.h to 1, flash will be locked at a maximum lock level the selected flash supports based on the configured flash layout. For details of Flash BP locking principle and flash layout configuration, please refer to RTL8762E Memory User Guide.

It should be notice that not all flash used by RTL8762E supports proportional block protect (0 ~ 1/2 ~ 1/4…). GD25WD80C, for instance, has a least block protect portion of 768KB, bigger than half. Flash like this could only be locked all or none in driver.

Table1. GD25WD80C Protected area size

Status Register Content			Memory Content			
BP2	BP1	BP0	Blocks	Addresses	Density	Portion
0	0	0	NONE	NONE	NONE	NONE
0	0	1	Sector 0 to 253	000000H-0FDFFFH	1016KB	Lower 254/256
0	1	0	Sector 0 to 251	000000H-0FBFFFH	1008KB	Lower 252/256
0	1	1	Sector 0 to 247	000000H-0F7FFFH	992KB	Lower 248/256
1	0	0	Sector 0 to 239	000000H-0EFFFFH	960KB	Lower 240/256
1	0	1	Sector 0 to 223	000000H-0DFFFFH	896KB	Lower 224/256
1	1	0	Sector 0 to 191	000000H-0BFFFFH	768KB	Lower 192/256
1	1	1	All	000000H-0FFFFFH	1024KB	ALL

Figure 4-2 Flash Block Protect for GD25WD80C

APIs are as follows:

- **bool flash_set_block_protect_locked(uint8_t bp_lv);**
- **bool flash_sw_protect_unlock_by_addr_locked(uint32_t unlock_addr, uint8_t *old_bp_lv);**
- **bool flash_get_block_protect_locked(uint8_t *bp_lv);**

5 Power Saving

Power mode of Flash is mainly divided into three scenarios: working, standby and Power Down. The power consumption of the working mode is generally about 10 mA, while the power consumption of standby mode is usually about 10uA order of magnitude. The power consumption of the Power Down mode is even lower, sometimes less than 1 uA.

Flash automatically enters the standby mode without any access, and automatically enters the working state when it needs to be accessed again. In order to enter the Power Down mode, a specific command has to be called. Most Flash use the command 0xB9 to enter the Power Down mode, and use the command 0xAB to exit the Power Down mode. But MXIC toggles the #CS pin to wake up flash.

After flash enters low power mode, it is dangerous to receive commands except for the exit DP command (0xAB). Because flash only accepts wakeup command to exit lower power mode, other commands will be ignored, while flash controller may step into an infinite loop waiting for response from flash.

In order to avoid the risk of abusing the Flash Power Down pattern, the system's DLPS mechanism has been added to the Flash Power Down mode control. When entering DLPS, the instruction automatically makes Flash enter the Power Down mode and wakes up when out of the DLPS.

6 XIP

If the remaining RAM space is sufficient, APP code can be directly executed on RAM, which is conducive to improving performance and reducing power consumption. However, if the APP code is so large that the remaining RAM space is not enough, some or all of the APP codes need to be executed on flash. Configuration of XIP on RTL8762E is as follows.

1. Macro FEATURE_RAM_CODE (configured in mem_config.h): when configured to 1, any code without section modification will be excecuted on RAM. On the contrary, when configured to 0, any code without section modifiers by default will be executed on flash.

2. Section modification (reference app_section.h)
   1) APP_FLASH_TEXT_SECTION: The specified code is executed on flash.
   2) DATA_RAM_FUNCTION: The specified code is executed on RAM. If RAM space is insufficient, you

should give higher priority to implementing time sensitive code on RAM to ensure efficiency.

3. Scene switching (reference app_section.h and overlay_mgr.h): APP development first divides the different scenarios according to the needs and defines the loading scene information table, then manually modifies the different scene code with different section keywords, and finally calls the loading function load_overlay in the place where the scene is switched. At present, RT8762D SDK supports the following three scenarios. But APP can expand according to certain principles.

```
#define OVERLAY_SECTION_BOOT_ONCE    __attribute__((section(".app.overlay_a")))
#define OVERLAY_B_SECTION            __attribute__((section(".app.overlay_b")))
#define OVERLAY_C_SECTION            __attribute__((section(".app.overlay_c")))
```

4. Two ways to improve the efficiency of XIP execution code
   1) Enable cache: Configure macro SHARE_CACHE_RAM_SIZE to 0K Bytes or 4. Cache size is 8K Bytes or 4K Bytes respectively. Please reference the section Cache in *RTL8762E Memory User Guide* for more details.
   2) Flash is switched to 2-bit mode or 4-bit mode by calling flash_try_high_speed.

RTL8762E can access flash with auto mode as accessing RAM through SPIC, and execute code directly on SPI Flash. However, the operation of accessing flash in user mode is not atomic. Once the accessing is interrupted by higher priority tasks or interrupts, it is possible to cause flash access error. To ensure atomicity of flash operations in user mode, XIP should follow the following restrictions and precautions.

Accessing flash operations in user mode require calling APIs with the "_locked" suffix to ensure a critical area protection. If the time of the flash operation is too long, such as writing a large number of data at one time, it is not suitable for the critical zone protection, and can be split into a small amount of data written by a few times.

If the time-critical interrupt needs to be processed, it is also necessary to ensure that the ISR (interrupt service routine) itself cannot be XIP, and the ISR also prohibits accessing to the flash. Besides, the ISR can’t cause task switch to a XIP task.

7 Other Flash APIs

- **uint32_t flash_ioctl(uint32_t cmd, uint32_t param_1, uint32_t param_2);**

Apart from read/write/erase/block protect and bit mode switch, RTL8762E also supports flash_ioctl api for all sorts of using. Here are some frequent-used commands:

1）flash_ioctl_get_size_main (0x0003)，get current flash size；

2）flash_ioctl_set_seq_trans_enable (0x1008)，set auto mode sequential transfer, enable when param_1 is 1, disable when 0；

3）flash_ioctl_app_base (0x5000)，get base address of APP；

4）flash_ioctl_exec_flash_sw_reset (0x3005)，execute software reset；

5）flash_ioctl_get_curr_bit_mode (0x0005)，get current flash bit mode；

6）flash_ioctl_get_rdid (0x0006)，get current flash ID。

- **bool flash_auto_dma_read_locked(T_FLASH_DMA_TYPE dma_type, FlashCB flash_cb, uint32_t src_addr, uint32_t dst_addr, uint32_t data_len);**

  High speed read api with auto mode and DMA.

- **bool flash_auto_seq_trans_dma_read_locked(T_FLASH_DMA_TYPE dma_type, FlashCB flash_cb, uint32_t src_addr, uint32_t dst_addr, uint32_t data_len);**

  High speed read api with auto mode and DMA, with sequential transfer enable, therefore faster than flash_auto_dma_read_locked.