

# **RTL8762D Flash User Guide**

**V1.0**

**2020/06/05**

## Revision History

Date	Version	Comments	Author	Reviewer
2020/06/05	V1.0	First release version	Yueming	Serval

Realtek Confidential

## Contents

Revision History .....	2
Figure List .....	4
1 Introduction .....	5
2 Flash AVL .....	5
3 Basic Operations.....	5
3.1 Flash API.....	5
3.1.1 Read .....	5
3.1.2 Write .....	6
3.1.3 Erase .....	6
3.2 Access Mode .....	6
4 Bit Modes .....	7
4.1 Three Bit Modes.....	7
4.2 Bit Mode Switch.....	7
5 Software Block Protect.....	8
6 Power Saving.....	9
7 XIP.....	10

## Figure List

Figure 5-1 Flash block protect.....	9
-------------------------------------	---

Realtek Confidential

# 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.

RTL8762D explores external flash, providing plenty of choices for users. For basic functions like single mode read, write and erase, multiple choices are available. However, RTL8762D flash driver also provides some advanced functions. For example, quad mode, deep power-down and flash block protect. These functions doesn't support all flash, therefore, Realtek releases and updates Approved Vendor List for approved flash. It is strongly suggested that customers choose flash from AVL to guarantee their functions.

RTL8762D supports executing code (XIP) on SPI Flash, but there are also some limitations and precautions.

## 2 Flash AVL

RTL8762D explores external flash with SPI interface. Users could choose different flash with different vendors for their own needs. But with so many flash vendors, flash has various functions and limitations, even with same vendors. Therefore, Realtek provides AVL for approved flash. Please refer to RTL8762D AVL.

## 3 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's 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.

### 3.1 Flash API

#### 3.1.1 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.

◆ **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.

### 3.1.2 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.

### 3.1.3 Erase

◆ **bool flash\_erase\_locked(T\_ERASE\_TYPE type, uint32\_t addr);**

flash\_erase\_locked supports 3 erase types (T\_ERASE\_TYPE), FLASH\_ERASE\_CHIP (erase whole chip), FLASH\_ERASE\_SECTOR (4K bytes) and FLASH\_ERASE\_BLOCK (64K bytes).

## 3.2 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 RTL8762D 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, RTL8762D supports 16KB cache. Flash controller supports two sets of mapped address spaces for at most 32MB size, corresponding to cache address space [0x00800000, 0x02800000) and non-cache address space [0x04800000, 0x06800000). When cache is enabled (the macro "SHARE\_CACHE\_RAM\_SIZE" for mem\_config.h is configured to be 8K or 16K), 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 0x04800000, 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.

## 4 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#)

### 4.1 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. That is one of the reasons why we need AVL.

### 4.2 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);
```

## 5 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. RTL8762D only supports locking from lower address of flash.



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

**Figure 5-1 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. RTL8762D 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 RTL8762D Memory User Guide.

## 6 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.

## 7 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 RTL8762D is as follows.

1. Macro `FEATURE_RAM_CODE` (configured in `mem_config.h`): when configured to 1, any code without section modification will be executed 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 8K Bytes or 0. Cache size is 8K Bytes or 16K Bytes respectively.
- 2) Flash is switched to 2-bit mode or 4-bit mode by calling flash\_try\_high\_speed.

RTL8762D 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.