FTL NAND Media Driver for Winbond W25N01G User Guide

Version 1.20

For use with FTL NAND Media Driver for Winbond W25N01G module versions 1.04 and above

Date: 11-Aug-2017 16:27

Table of Contents

1 System Overview

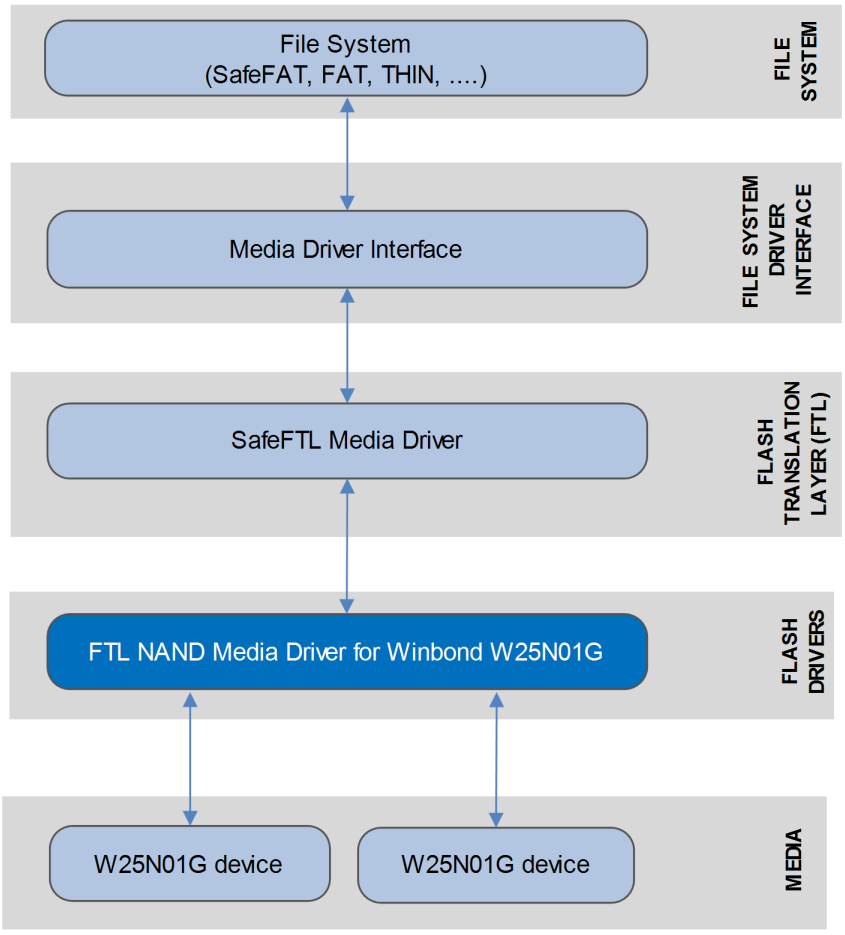
1.1 Introduction

This guide is for those who want to use HCC's FTL NAND Media Driver for Winbond W25N01G flash devices in their system. These are produced by Winbond Electronics Corp. This guide covers all aspects of configuration and use.

The FTL conforms to the *HCC Media Driver Interface Specification*. This driver conforms to the FTL low level driver interface specification described in the *SafeFTL User Guide*.

The media driver provides an interface for a file system to read from and write to NAND Flash storage devices. A single media driver can support one or more physical media, each of these being represented as a different drive at the media driver interface. The file system handles all drives identically, regardless of their internal design features.

The diagram below shows a typical system architecture including a file system, media driver and media. As an example, this shows two devices.



Note the following:

- The file system can be any HCC file system that addresses logical sector arrays (including SafeFAT, FAT, and THIN).
- The Flash Translation Layer (FTL) is the SafeFTL media driver. This has its own manual.
- The NAND flash driver is written specifically for the NAND flash controller (integrated with the W25N01G microcontroller) and the specific NAND flash array used.
- The driver uses HCC's standard SPI interface so can be easily integrated with any microcontroller with SPI.

1.2 Feature Check

The main features of the media driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Conforms to the low level NAND flash interface defined by HCC's SafeFTL.
- Supports multiple Winbond W25N01G flash drives.
- Uses HCC's standard SPI interface so integrates easily with any microcontroller with SPI.

1.3 About Winbond NAND Flash

This table summarizes the properties of the Winbond W25N01GVxxIG/IT device types:

Character(s)	Description
W	Winbond company prefix.
25N	Product family Serial SLC NAND.
01G	Density (1Gb).
V	Supply voltage.
xx	Package type: • ZE = 8 pad WSON. • TB = 24 ball TFBGA.
I	Temperature = industrial.
G	By default BUF is 1 after power up.
T	By default BUF is 0 after power up.

Also note the following:

- This device requires 1 bit ECC correction per 528 bytes.
- The device has an internal ECC calculator that the driver uses.
- All W25N0xG devices have a page size of 2048 bytes with a 64 byte spare area.

Device Compatibility

The driver is tested with W25N01G but is also fully compatible with W25N02G and W25N04G devices.

1.4 Packages and Documents

Packages

The table below lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
media_drv_ftl_base	The base SafeFTL package.
media_drv_ftl_nand_w25n_spi	The media driver package described in this document.
psp_template_spi	The SPI Platform Support Package (PSP).

Documents

For an overview of HCC file systems and flash management technologies, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Media Driver Interface Guide

This document describes the HCC Media Driver Interface Specification.

HCC SafeFTL User Guide

The user guide for SafeFTL.

HCC FTL NAND Media Driver for Winbond W25N01G User Guide

This is this document.

1.5 Change History

This section describes past changes to this manual.

- To download earlier manuals, see Archive: FTL NAND Media Driver for Winbond W25N01G User Guide.
- For the history of changes made to the package code itself, see History: media_drv_ftl_nand_w25n_spi.

The current version of this manual is 1.20. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.20	2017-08-11	1.04	Corrected *Packages* list.
1.10	2017-06-27	1.04	New *Change History* format.
1.00	2015-12-11	1.03	First online version.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_ftldrv_w25n.h**. is the only file that should be included by an application using this module. For details of the single API function, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_ftldrv_w25n.h** contains all the configurable parameters of the system. Configure these as required.

2.3 Source Code File

The file **src/media-drv/ftl/drivers/nand/winbond/w25n.c** holds the source code for the media driver. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_ftldrv_w25n.h** contains the version number of this module. This version number is checked by all modules that use a module to ensure system consistency over upgrades.

3 Configuration Options

Set the system configuration options in the file **src/config/config_ftldrv_w25n.h**. This section lists the available configuration options and their default values.

W25N_SPI_UID

The SPI unit ID. The default is 0.

W25N_SPI_BAUD_RATE

The frequency for SPI. The default is 5000000.

W25N_FREE_BLOCK_MIN

The number of free blocks. The value cannot exceed MDRIVER_FTL_MAX_FREE_BLOCKS. The default is 64.

The total number of management blocks is (W25N_FREE_BLOCK_MIN + maximum number of bad blocks). This depends on the device detected.

W25N_LOG_BLOCK_AVAILABLE

The number of log blocks. The value cannot exceed MDRIVER_FTL_MAX_LOG_BLOCK_AVAIL. The default is 5.

W25N_NUM_OF_DIF_MAPBLOCK

The number of blocks used for mapping in the system. The default is 2. The valid range is 1 to 16.

W25N_MAPBLOCK_SHADOW

The number of map shadow blocks. The default is 2. The value must not be below 1.

The system may be more efficient if more map shadow blocks are used, but each additional block reduces the number of free blocks in the system.

W25N_RESERVED_BLOCKS

The number of reserved blocks, the blocks at the start of the flash area that driver should not use. The default is 0.

W25N_WEAR_STATIC_LIMIT

The maximum value that the difference between the maximum and minimum wear count can be. The default is 1024.

W25N_WEAR_STATIC_COUNT

The number of merge operations after which static wear checking must be run. The default is 128.

W25N_LL_TEST

Set this to 1 to enable low level driver raw testing mode. The default is 0.

3.1 Restriction

The following value must be 8 or greater:

```
W25N_FREE_BLOCK_MIN - ( W25N_NUM_OF_DIF_MAPBLOCK * W25N_MAPBLOCK_SHADOW + 1 ) -
W25N_LOG_BLOCK_AVAILABLE
```

4 Application Programming Interface

This section describes the single Application Programming Interface (API) function and the structure it uses.

When the media driver is used:

1. The file system calls the media driver's **ftldrv_w25n_init()** function.
2. **ftldrv_w25n_init()** returns a pointer to a *t_ftl_driver* structure containing a set of functions for accessing the media driver.

4.1 ftldrv_w25n_init

This function initializes the driver for this device.

This is called automatically from SafeFTL, using its table of flash drives. See the *HCC SafeFTL User Guide* for details.

Format

```
t_ftl_ret ftldrv_w25n_init (
    uint32_t           drvnum,
    t_ftl_driver * *   pps_ftl_driver )
```

Arguments

Argument	Description	Type
drv_num	The number of the drive to initialize. The first drive is 0.	uint32_t
pps_ftl_driver	On return, a pointer to a *t_ftl_driver* structure defining the interface to that driver.	t_ftl_driver * *

Return values

Return value	Description
0	Successful execution.
1	Operation failed.

4.2 SafeFTL Flash Drive Structure Example

SafeFTL uses a flash drive structure containing all the available flash drives. Each available flash driver must have an entry in this table, specifying its initialization function and the parameter to be passed to it in that function. The flash drives are numbered from 0 to (MDRIVER_FTL_MAX_DRIVE-1). The index to this table is used to reference the flash drive.

This structure is held in the main SafeFTL package's **src/config/config_mdriver_ftl.c** file.

The following example shows how a W25N01G drive would appear in this structure. In this case it is the only NAND drive, followed by two NOR drives.

```
t_ftldrive_init  as_ftldrive_init[MDRIVER_FTL_MAX_DRIVE] =
{
  { ftldrv_w25n_init, 0U }
  , { ftl_nor_init, 0U }
  , { ftl_nor_init, 1U }
};
```

4.3 Error Codes

The possible return codes are shown in the table below:

Code	Value	Description
LL_OK	0	Successful execution.
LL_ERASED	1	Page is empty.
LL_ERROR	2	Other error.

4.4 t_ftl_driver

The **ftldrv_w25n_init()** function returns a pointer to this *t_ftl_driver* structure:

Element	Type	Description
user_data	uint32_t	User-defined data.
pf_getphy	(* pf_getphy)	Pointer to **getphy()** function.
pf_read	(* pf_read)	Pointer to **read()** function.
pf_readpart	(* pf_readpart)	Pointer to **readpart()** function.
pf_write	(* pf_write)	Pointer to **write()** function.
pf_writedouble	(* pf_writedouble)	Pointer to **writedouble()** function.
pf_erase	(* pf_erase)	Pointer to **erase()** function.
pf_isbadblock	(* pf_isbadblock)	Pointer to **isbadblock()** function.
pf_readonebyte	(* pf_readonebyte)	Pointer to **readonebyte()** function.

5 Integration

This section describes all aspects of the module that require integration with your target project. This includes porting and configuration of external resources.

5.1 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The module makes use of the following standard PSP function:

Function	Package	Element	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.

The module makes use of the following standard PSP SPI functions. These are described in the *HCC SPI Driver PSP User Guide*.

Function	Package	Element	Description
psp_spi_init()	psp_base	psp_spi	Initializes the SPI port.
psp_spi_cs_hi()	psp_base	psp_spi	Sets chip select high.
psp_spi_cs_lo()	psp_base	psp_spi	Sets chip select low.
psp_spi_set_baudrate()	psp_base	psp_spi	Sets the baudrate.
psp_spi_rx()	psp_base	psp_spi	Receives a number of bytes.
psp_spi_tx1()	psp_base	psp_spi	Transmits one byte.