Application Report
SWRA657–June 2020

SimpleLink™ Wi-Fi® CC3100, CC3200 Serial Flash

Michael Reymond

ABSTRACT

This application report provides important guidelines and best practice design techniques to consider when choosing and embedding a serial flash paired with the CC3100/CC3200TM devices. It also describes the file system, along with guidelines and considerations for system designers working with the CC3100/CC3200TM file system.

Contents

List of Figures

List of Tables

Trademarks

SimpleLink is a trademark of Texas Instruments.
Wi-Fi is a registered trademark of Wi-Fi Alliance.
All other trademarks are the property of their respective owners.

1 Introduction

Many embedded systems contain a serial flash component to store firmware, configuration files and user data for usage by a microcontroller/processor. The processor sporadically writes data into this serial flash to update its contents. Inclusion of serial flash memory poses unique challenges for system designers.

- A typical serial flash ensures a data endurance of 100K write cycles per sector and 20 years data retention. The write endurance and data retention characteristics must to be taken into account by the application developer.
- The serial nature of reads and writes results in long access times and this raises the challenge of maintaining a stable system supply voltage for the duration of the access.

2 How File System Content Gets to the Serial Flash

Before diving into the file system structure, it is highly important for designers to understand how content is created on the serial flash.

There are several ways for files to be created on the serial flash. Table 1 describes the various options.

Table 1. File Creation on Serial Flash

Application/Tool	Device State	Content Delivery Channel	Content Type	Package/Example
Uniflash	Bootloader	UART	User, system and configuration files	SimpleLink Wi-Fi CC3100, CC3200 UniFlash User's Guide (SWRU558)
SimpleLinkTM file system API	Operational	UART/SPI	User files	• File operations – SDK • OTA • Host programming
SimpleLink non file system API	Operational	UART/SPI	System and configuration files	Any SDK example using the SimpleLink host driver
SimpleLink Device internals	Operational	Device internals	N/A	

3 File System Guidelines

As the file system may be used by customers for their own purposes, it is essential to understand the constraints and recommendations so the file system is designed and maintained properly.

The guidelines are:

- The minimum flash resolution is one 4KB block
- The file system itself requires three blocks for a total of 12KB
- There are two reserved blocks that cannot be used
- The total number of files is limited to 128 files, including system and configuration files
- The maximum size for a file is 1MB
- The maximum size for the serial flash is 16MB
- Each file will consume at a minimum:
  - One block (4KB) for file without fail-safe support
  - Two blocks (8KB) for file with fail safe support (double the original size)

- It is required to set the maximum size attribute upon file creation (the file system reserves space). In this case, the actual size of the file is irrelevant as file system occupies the allocated space.
- The file system reserves space per file at file creation. It is required to specify the maximum size of a file at creation, that maximum size is the space occupied by the file in serial flash, regardless of the actual size of the contents.
- The maximum file size and space reserved by the file system for a file cannot be changed after a file is created.
- File attributes (currently, only fail-safe) cannot be modified after the file has been created
- There is no fragmentation in the file system. This means that an existing file which is removed will leave a hole in memory. This hole may be reallocated by the file system if a newly created file can fit.
- When file is created, the minimal size memory hole that can hold the file is allocated to it.
- The actual size of a file may increase depending on its size. For more information on this mapping, see Section 4.
- Each file has a header while enlarges the actual size of the file. This overhead is 400 bytes. If the file size is close to a multiple of the 4KB block size, this may result in an additional block being allocated. For example, a file with an actual size of 4000 bytes would result in two blocks being allocated one the header is taken into account as 4000 + 440 bytes > 4096 bytes. For more information, see Section 5.
- Some system/configuration files may be created internally by the device. For more information, see Section 5.
- Some system and configuration files may be created implicitly by the host processor as a result of invoking non file-system APIs. For more information, see Section 5.

4 User File Mathematics

To make use of the file system to store files, it is essential to be able to accurately calculate the occupied memory space per file. The total occupied size on flash is a function of the file content length (or the maximum size upon creation), file attributes and file system metadata.

Figure 1 describes the process for calculating how much memory is actually consumed on the serial flash.

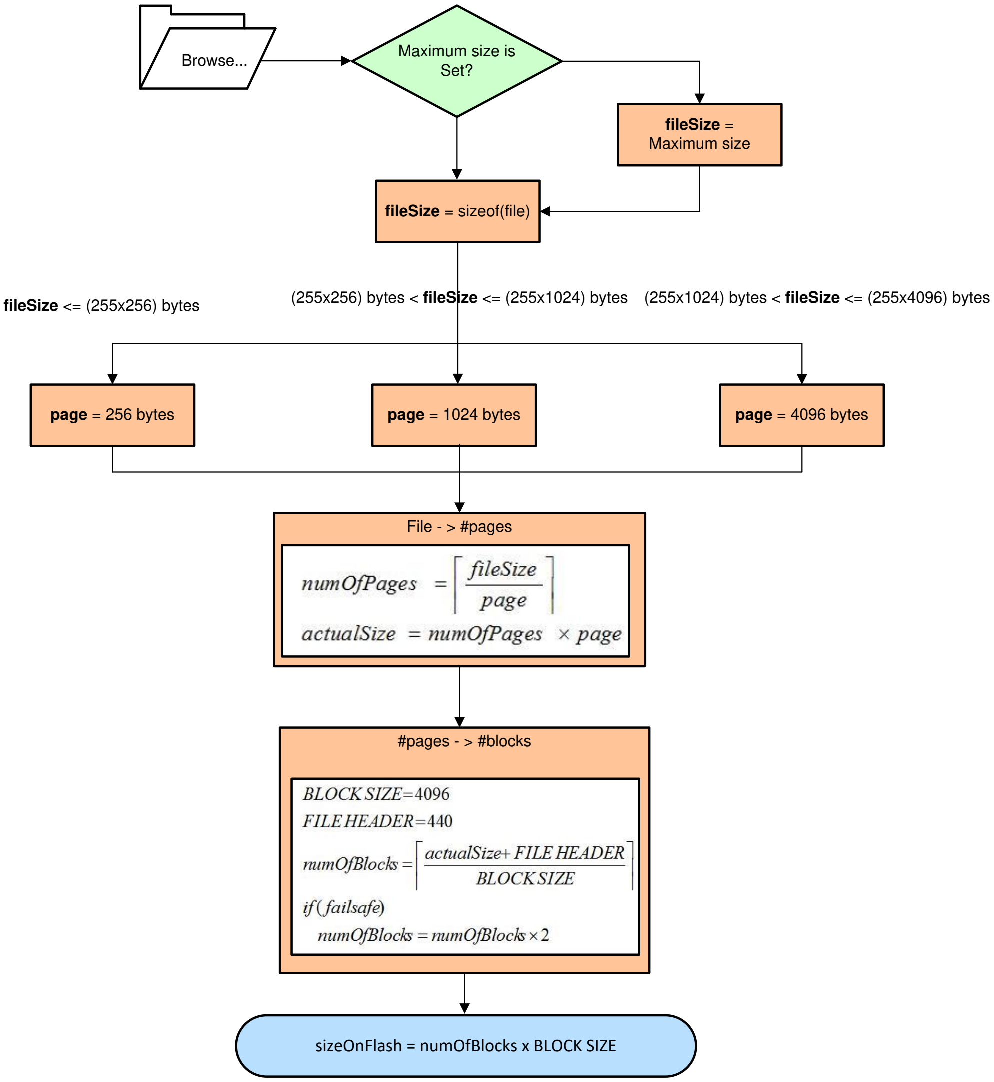
Browse...
Maximum size is Set?
fileSize = Maximum size
fileSize = sizeof(file)
fileSize <= (255x256) bytes
(255x256) bytes < fileSize <= (255x1024) bytes
(255x1024) bytes < fileSize <= (255x4096) bytes
page = 256 bytes
page = 1024 bytes
page = 4096 bytes
File - > #pages
numOfPages = ⌈fileSize / page⌉
actualSize = numOfPages × page
#pages - > #blocks
BLOCK SIZE=4096
FILE HEADER=440
numOfBlocks = ⌈(actualSize+ FILE HEADER) / BLOCK SIZE⌉
if(failsafe)
numOfBlocks = numOfBlocks × 2
sizeOnFlash = numOfBlocks x BLOCK SIZE

Figure 1. File Size Calculations

Table 2. File Size Allocation on Flash

File Size (bytes)	Page Size	Round to Page	Add Header and Round to Block	Allocated Size via sl_FsGetInfo()
10	256	256 (1 page)	4096 (1 block)	3656
250000	1024	250880 (245 pages)	253952 (62 blocks)	253512
1000000	4096	1003520 (245 pages)	1007616 (246 blocks)	10007176

5 System and Configuration Files

Configuration and system files can be implicitly created by application upon calling specific APIs in addition to being created internally by the device. Since these files are essential for proper device operation, it is important that you understand the meaning of each file and when it is required. Failing to preserve enough space for system/configuration files may cause system unexpected behavior.

5.1 Description

Table 3 describes in details the meaning of each system or configuration file.

Table 3. System File Description

Artifact	Description
FAT	File allocation table. Created upon format operation, either from Uniflash or via Image Programming. FAT is device unique so serial flash devices cannot be cloned and used on another SimpleLink device
Reserved blocks	Reserved blocks for future use
/sys/servicepack.ucf	Holds firmware patches to all internal cores, Network processor, MAC layer and PHY layer.
/tmp/phy.cal	Calibration file created automatically by the device. Calibration is performed sparingly, typically when coming out of Hibernate and only if temperature has changed by more than 20°C or upon power cycle
/tmp/fcon.frm	Fast connect frame. Holds all beacon information of the last connected access point. Created upon successful connection to an access point
/tmp/fcon.ssid	Fast connect profile. Holds selected profile information of the last connected access point, as if a profile is added by the application through API. Created upon successful connection to an access point.
/tmp/table.arp	ARP table file. This file holds up to 8 entries. The table is updated upon ARP response reception to the device IP address
/sys/date_time.cfg	Holds the system time and date. Mainly used with to validate expiry time of certificates during SSL connections
/sys/rxfltr.ini	Holds all RX filtering information. The Rx-filters module enables you to simply define a set of rules that determine which of the received frames are transferred by the SimpleLink device to the host, and which frames are dropped. The Rx-filters can be activated during AP-connection and during promiscuous mode
/sys/infoele.cfg	Holds user defined information element. The information element is transmitted when the device is either in AP mode or Peer2Peer group owner mode. The size is limited to 252 bytes
/sys/macadd.bin	Holds the user defined MAC address of the device. If this file exists, the device internal MAC address is bypassed
/sys/stacfg.ini	Holds device configuration when it is either in station mode or Peer2Peer client mode. The list of APIs related to stacfg.ini can be found under Host Driver mapping
/sys/ap.cfg	Holds device configuration when in access point mode. The list of APIs related to ap.cfg can be found under Host Driver mapping
/sys/p2p.cfg	Holds device configuration when Peer2Peer mode (client or group owner). The list of APIs related to p2p.cfg can be found under Host Driver mapping
/sys/mode.cfg	Holds device general configuration, unbound to device mode. The list of APIs related to mode.cfg can be found under Host Driver mapping
/sys/pref.net	Holds up to 7 stored profiles. Profiles may be stored explicitly from user application or implicitly during Smart Config provisioning. SimpleLink device scan the stored profile list only if connection policy is set to Auto mode
/sys/pmcfg.ini	Holds the power policy configuration of the device. The list of APIs related to pmcfg.ini can be found under Host Driver mapping
/sys/ipcfg.ini	Hold IP configuration of the device. The list of APIs related to ipcfg.ini can be found under Host Driver mapping
/sys/devname.cfg	Holds the device name and URN. The list of APIs related to devname.cfg can be found under Host Driver mapping
/sys/dhcpsrv.cfg	Holds DHCP server information. The list of APIs related to dhcpsrv.cfg can be found under Host Driver mapping
/sys/httpsrv.cfg	Holds HTTP server information. The list of APIs related to httpsrv.cfg can be found under Host Driver mapping
/sys/mdns.cfg	Holds mDNS client information. The list of APIs related to mdns.cfg can be found under Host Driver mapping

Table 3. System File Description (continued)

Artifact	Description
/sys/smartconfigkeys.cfg	Holds Smart Config keys in case the provisioning procedure is secured (access point credentials are encrypted during provisioning).

5.2 Memory Consumption

Table 4 illustrates the memory consumption for each of the system components.

Table 4. System File Memory Consumption

Filename/Artifact	Description	Fail Safe	#blocks
FAT	File allocation table. Created upon format operation.	N/A	3
Reserved blocks	Reserved blocks for the system	N/A	2
Total			**5**
/sys/servicepack.ucf	Firmware patches	yes	66
/tmp/phy.cal	PHY calibration	yes	10
/tmp/fcon.frm	Fast Connect frame (beacon info)	yes	2
/tmp/fcon.ssid	Fast Connect profile	yes	2
/tmp/table.arp	Arp Table	yes	2
/sys/date_time.cfg	Date Time	yes	2
/sys/rxfltr.ini	Rx Filters	yes	2
/sys/infoele.cfg	Info Element file	yes	2
Total			**88**
/sys/macadd.bin	MAC address	yes	2
/sys/stacfg.ini	Station configuration	yes	2
/sys/ap.cfg	AP configuration	yes	2
/sys/p2p.cfg	Peer2Peer configuration	yes	2
/sys/mode.cfg	WLAN Mode	yes	2
/sys/pref.net	Preferred networks	yes	4
/sys/pmcfg.ini	Power Management	yes	2
/sys/ipcfg.ini	IP configuration	yes	2
/sys/devname.cfg	Device Name	yes	2
/sys/dhcpsrv.cfg	DHCP Server configuration	yes	2
/sys/httpsrv.cfg	HTTP Server configuration	yes	2
/sys/mdns.cfg	mDNS configuration	yes	2
/sys/smartconfigkeys.cfg	SmartConfig Keys	yes	2
Total			**28**

5.2.1 CC3200 Use Case

CC3200 is a special case since its image (application code running on the internal processor) should be taken into account as well. The occupied space depends on the over-the-air (OTA) scheme. Table 5 illustrates two OTA schemes: MCU image with fail-safe option and OTA as taken from the OTA SDK add-on.

Table 5. CC3200-Specific System File Memory Consumption

Filename/Artifact	Description	Fail Safe	No blocks
/sys/mcuimg.bin	MCU application image running on the internal Cortex M4 host processor	yes	128
Total			**128**
MCU boot info	Boot information for image selection	no	1
/sys/mcuimg.bin (MCU application bootloader)	MCU application for image selection	no	1
MCU image 1	1st image – the default image	no	64
MCU image 2	2nd image	no	64
MCU image 3	3rd image	no	64
Total			**194**

5.2.2 Host Driver Mapping

As described earlier, some files are created internally and some can also be created implicitly by invoking host driver APIs.

It is mandatory to understand the mapping so you are able to design your memory budget such that enough space is preserved for all required system/configuration components.

Table 6. Host Driver File Mapping

Filename/Artifact	Created by API()	When Exiting	Write Period	No Blocks
FAT	N/A	When Exit Reset When Exit HIB	At format After every file write or deletion Every Added/Remove File	3
Reserved blocks	N/A	N/A	N/A	2
/sys/mcuimg.bin	Refer to OTA	When Exit Reset When Exit HIB	When updating APP	OTA dependent
/sys/servicepack.ucf	Refer to OTA. sl_FsOpen(),sl_FsWrite() and sl_FsClose()	When Exit Reset When Exit HIB	User Initiated	66
/tmp/phy.cal	N/A	When Exit Reset When Exit HIB	At least once, Upon temperature change when coming out of hibernate or upon power cycle	10

/tmp/fcon.frm	sl_WlanConnect()
	sl_WlanProfileAdd()
	sl_WlanSmartConfigStart()
		Upon successful AP connection	When AP changes	2

/tmp/fcon.ssid	sl_WlanConnect()
	sl_WlanProfileAdd()
	sl_WlanSmartConfigStart()
		Upon successful AP connection	When AP changes	2

/tmp/table.arp	N/A	Upon successful AP connection	ARP response to the device IP address	2
/sys/date_time.cfg	sl_DevSet(SL_DEVICE_GENERAL_CONFIGURATION, SL_DEVICE_GENERAL_CONFIGURATION_DATE_TIME, …)	When Exit Reset When Exit HIB User Initiated	User Initiated	2
/sys/rxfltr.ini	sl_WlanRxFilterAdd() with sl_WlanRxFilterSet(SL_ENABLE_DISABLE_RX_FILTER,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2

Table 6. Host Driver File Mapping (continued)

Filename/Artifact	Created by API()	When Exiting	Write Period	No Blocks
/sys/infoele.cfg	sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID,WLAN_GENERAL_PARAM_OPT_INFO_ELEMENT,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2
/sys/macadd.bin	sl_NetCfgSet(SL_MAC_ADDRESS_SET,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2

/sys/stacfg.ini	sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID, WLAN_GENERAL_PARAM_OPT_SCAN_PARAMS,…)
	sl_WlanPolicySet(SL_POLICY_CONNECTION, SL_CONNECTION_POLICY(),…)
	sl_WlanPolicySet(SL_POLICY_P2P, SL_P2P_POLICY(),…)
	sl_WlanPolicySet(SL_POLICY_SCAN,SL_SCAN_ENABLE,…)
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/ap.cfg	sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SSID,…
	sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_CHANNEL,…)
	sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_HIDDEN_SSID,…)
	sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_SECURITY_TYPE,…)
	sl_WlanSet(SL_WLAN_CFG_AP_ID, WLAN_AP_OPT_PASSWORD,…)
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/p2p.cfg	sl_WlanSet(SL_WLAN_CFG_P2P_PARAM_ID, WLAN_P2P_OPT_DEV_TYPE,…)
	sl_WlanSet(SL_WLAN_CFG_P2P_PARAM_ID, WLAN_P2P_OPT_CHANNEL_N_REGS,…)
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/mode.cfg	sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID, WLAN_GENERAL_PARAM_OPT_STA_TX_POWER,…)
	sl_WlanSet(SL_WLAN_CFG_GENERAL_PARAM_ID, WLAN_GENERAL_PARAM_OPT_AP_TX_POWER,…)
	sl_WlanSetMode()
	sl_EventMaskSet()
	sl_NetAppStart()
	sl_NetAppStop()
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/pref.net	sl_WlanProfileAdd()
	sl_WlanProfileDel()
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/pmcfg.ini	sl_WlanPolicySet(SL_POLICY_PM,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	4

/sys/ipcfg.ini	sl_NetCfgSet(SL_IPV4_STA_P2P_CL_STATIC_ENABLE,IPCONFIG_MODE_ENABLE_IPV4,…)
	sl_NetCfgSet(SL_IPV4_STA_P2P_CL_DHCP_ENABLE,IPCONFIG_MODE_ENABLE_IPV4,…)
	sl_NetCfgSet(SL_IPV4_AP_P2P_GO_STATIC_ENABLE,IPCONFIG_MODE_ENABLE_IPV4,…)
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/devname.cfg	sl_NetAppSet (SL_NET_APP_DEVICE_CONFIG_ID,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2
/sys/dhcpsrv.cfg	sl_NetAppSet(SL_NET_APP_DHCP_SERVER_ID,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2

Table 6. Host Driver File Mapping (continued)

Filename/Artifact	Created by API()	When Exiting	Write Period	No Blocks
/sys/httpsrv.cfg	sl_NetAppSet(SL_NET_APP_HTTP_SERVER_ID,…)	When Exit Reset When Exit HIB User Initiated	User Initiated	2

/sys/mdns.cfg	sl_NetAppSet(SL_NET_APP_MDNS_ID,…)
	sl_NetAppMDNSRegisterService()
	sl_NetAppMDNSUnRegisterService()
		When Exit Reset
		When Exit HIB
		User Initiated
			User Initiated	2

/sys/smartconfigkeys.cfg	Creation via Uniflash only Invocation via sl_WlanSmartConfigStart()	When Exit Reset When Exit HIB User Initiated	User Initiated	2

5.2.3 Minimum Flash Size

The minimum size of the serial flash is determined according to the memory space occupied by all system and configuration files.

As described earlier, you can either reserve the entire space or customize and reserve only the required space. In the latter case, extra caution should be taken since not preserving enough space may cause unexpected system behavior.

Table 7 marks the full memory consumption in case all system and configuration files are used.

Table 7. Flash Memory Consumption

	CC3100	CC3200
File system artifacts (FAT, reserved blocks)	20KB	20KB
Service Pack	264KB	264KB
System and Configuration	200KB	200KB
MCU scheme with fail-safe [1]	N/A	512KB
MCU scheme from OTA example [2]	N/A	776KB
Total	484KB	996KB/1260KB

[1] 996KB is for MCU scheme with fail-safe option whereas 1260 KB is for MCU scheme from OTA example where all MCU images are created with no fail-safe. The decision mechanism is implemented in host.

[2] If the “Over The Air” example provided by TI is used as a reference, then the MCU application binaries need not be created as “fail safe”, as the primary and back-up images are stored as distinct files in the file system.

You have to select a flash large enough to fit the application code and required data files. Note that flash devices are available in the sizes, 8 Mbit, 16 Mbit, 32 Mbit, 64 Mbit and 128 Mbit.

6 Implementing File System Features From Host Processor

6.1 Overview

You may find it frustrating that the SimpleLink host driver does not expose some file system functionality that are typically considered "built-in". These features include file listing, occupied/free space and file appending. The reason these features are not part of the file system stems from the fact that the file system was originally designed to serve only the internal cores of the device. Since it is impossible to add such features as patches, current production devices lack these services from the device. However, this section offers guidance on how to implement such features in the host application.

6.2 File Listing

6.2.1 Requirements

- Keep and maintain a list of all user filenames created from host
- Full list of all system and configuration filenames is required. For more information, see Section 5.
- If allocated, size per file is also required:
  - Keep and maintain for each user file its allocated size (either the size of the content or the maximum size argument)
  - Full list of all system and configuration file sizes is required. For more information, see Section 5.
- Knowledge of how to accurately calculate the occupied memory space per file. For more information, see Section 4.

6.2.2 Procedure

The procedure for listing the files and their respective allocated space is as follows:

1. Keep the following information for every user file created:
   a. Filename
   b. Calculate file size according to Section 4.
2. Loop over all user files and print:
   a. Filename
   b. Calculated size
3. Loop over all system/configuration files, invoke fsGetInfo() API and conclude:
   a. If return value is SL_FS_ERR_FILE_NOT_EXISTS, file does not exist and thus should not be printed. Else, the file exists and should be printed along with the allocated size
   b. If return value is SL_FS_ERR_TOKEN_IS_NOT_VALID, it means the file is secured but exists. This return value applies to the following three files:
      i. /tmp/fcon.frm
      ii. /sys/pref.net
      iii. /sys/smartconfigkeys.cfg

In this case, the file exists and should be printed along with the allocated size.

Figure 2 illustrates the procedure.

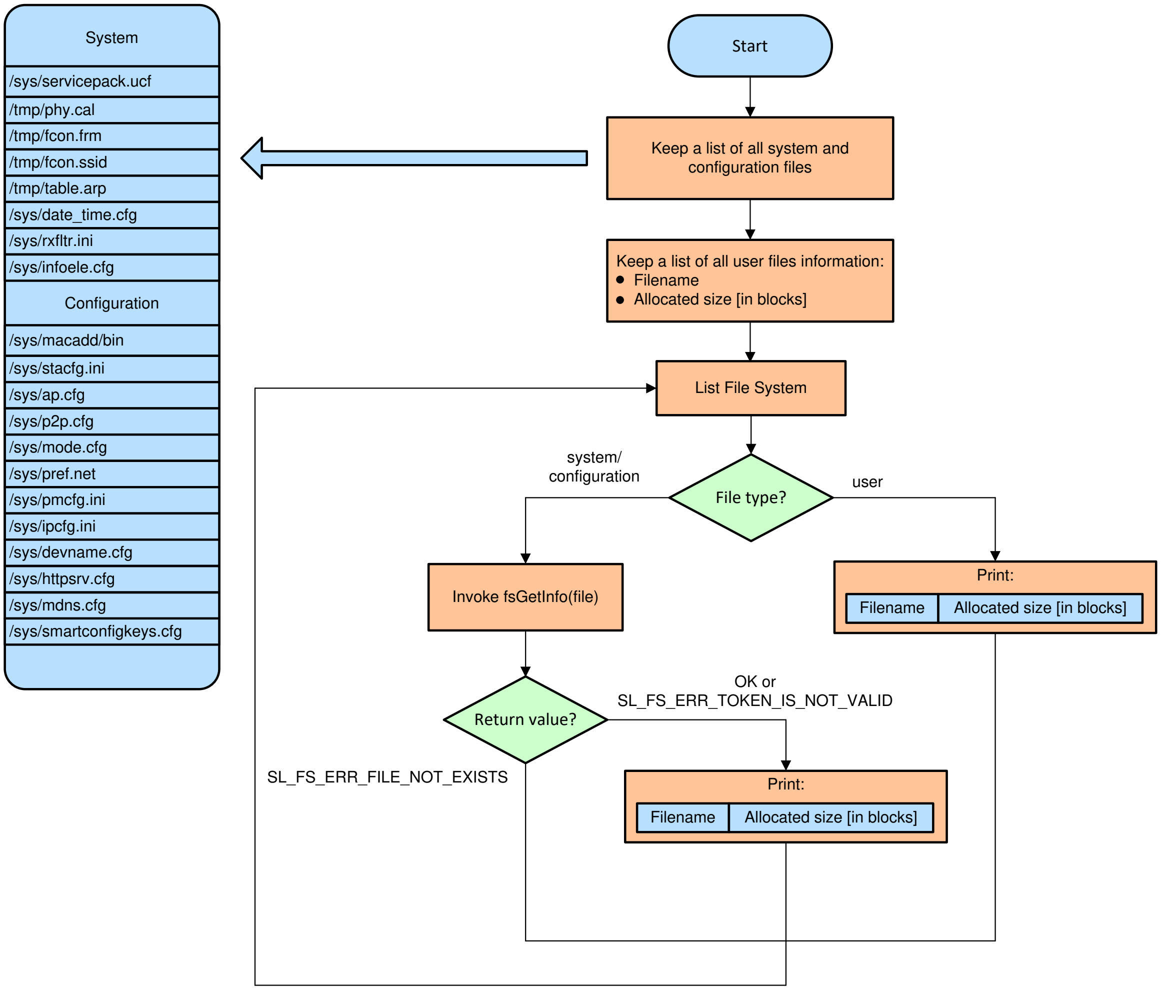

Figure 2. File Listing Flow

6.3 Free/Occupied Space

6.3.1 Requirements

- Keep and maintain a list of all user filenames created from host.
- Keep and maintain the allocated size (either the size of the content or the maximum size argument) for each user file.
- Knowledge of how to accurately calculate the occupied memory space per file. For more information, see Section 4.
- Full list of all system and configuration filenames and sizes are required. For more information, see Section 5.

6.3.2 Procedure

The procedure for calculating free/occupied space is as follows:

1. Initialize the free space to the size of the flash (as formatted by Uniflash) and the occupied space to 0. Units are blocks (of 4KB)
2. Account for FAT and the reserved blocks:
    a. Decrease free space by 5 blocks
    b. Increase occupied space by 5 blocks
3. Calculate the allocated size for every user file created. For more information, see Section 4.
4. Loop over all user files and for every file apply:
    a. Decrease free space by the allocated size in blocks
    b. Increase occupied space by the allocated size in blocks
5. Loop over all system/configuration files, invoke fsGetInfo() API and conclude:
    a. If return value is SL_FS_ERR_FILE_NOT_EXISTS, file does not exist and thus counters should remain, else, if no error is returned, followings should be updated:
        i. Decrease free space by the allocated size in blocks
        ii. Increase occupied space by the allocated size in blocks
    b. If return value is SL_FS_ERR_TOKEN_IS_NOT_VALID, it means the file is secured but exists. This return value applies to the following three files:
        i. /tmp/fcon.frm
        ii. /sys/pref.net
        iii. /sys/smartconfigkeys.cfg

In this case, the file exists and followings should be updated:

- Decrease free space by the allocated size in blocks
- Increase occupied space by the allocated size in blocks

Figure 3 illustrates the procedure.

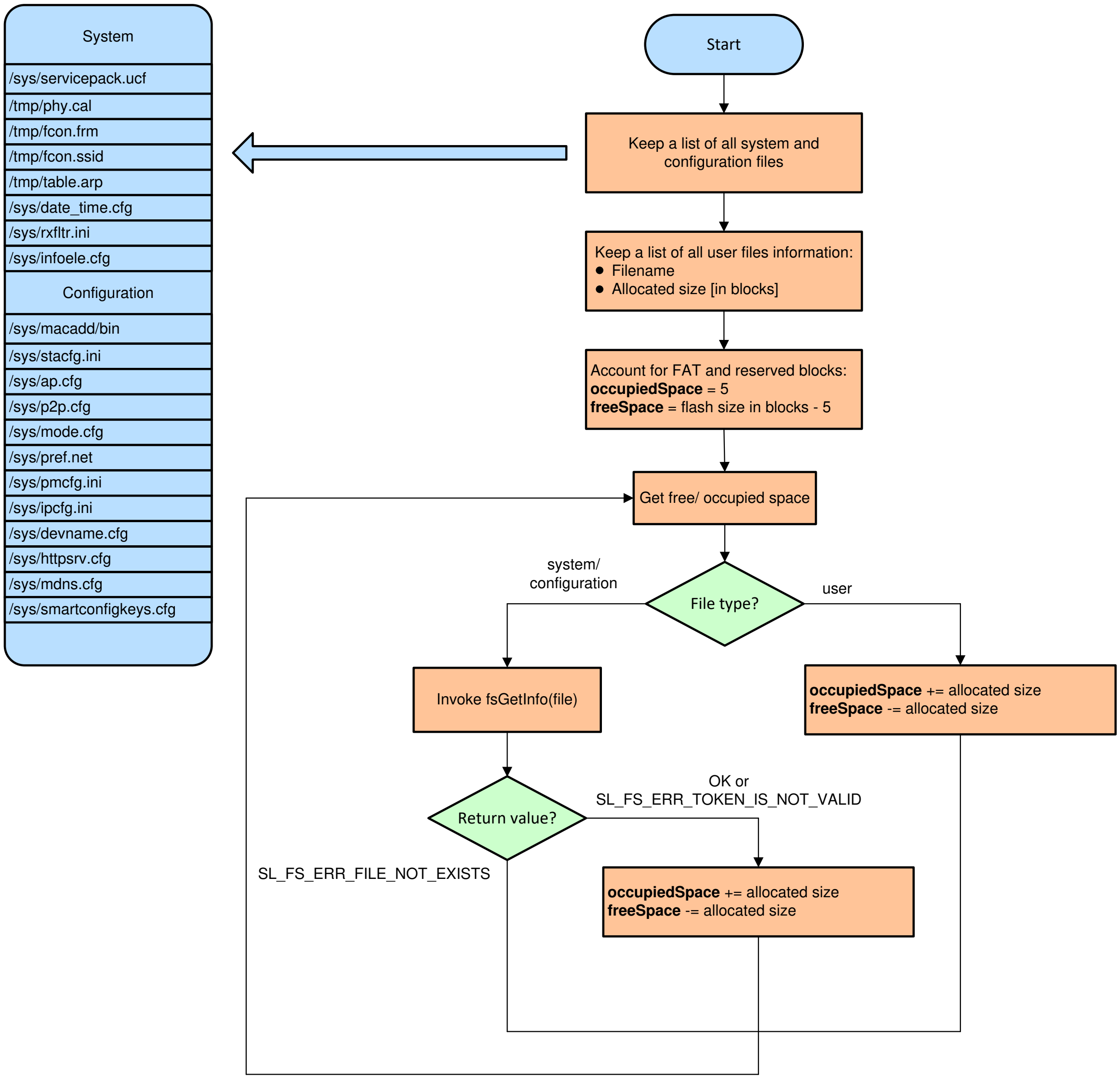

Figure 3. Free Space and Occupied Space Calculation

6.4 File Appending

6.4.1 Requirements

No preliminary requirements.

6.4.2 Procedure

There are several options to implement file appending from the host application. All options are based on read → modify → write.

The assumption is that the host processor does not have enough resources to read the entire file to its space so the serial flash needs to be used as a mediator. The proposed approach is to keep two files that hold the same copy but in different time instances. The update/append is applied to the “older” instance (based on the “newer” instance) and upon successful update/append, it becomes the “newer” instance. So practically this approach implements rollback option in host level and does not make use of the file system fail-safe option.

The procedure for file appending is as follows:

1. Create a file and write some content into it. The file would be denoted as file1.
2. Initialize a counter to file1 denoted as file1_counter and initialize it to 1.
3. The other instance is denoted as file2 and its counter file2_counter is initialized to 0. The “newer” file at this point is file1 and its counter indicates it (file1_counter> file2_counter).
4. 4. Now, if an update/append is required:
   a. Look for the “older” instance. If (file1_counter<file2_counter) file1 is the “older” instance, otherwise it is file2.
   b. Open the “newer” file for read.
   c. Open the “older” file for write.
   d. Read from the “newer” and write to the “older”. At this point, any power fail would not corrupt the file as the “newer” instance is opened for read.
   e. When reading from “newer” and writing to “older” is done, apply the followings:
      i. Update/append to “older” instance.
      ii. Close the “newer” instance.
      iii. Close the “older” instance.
      iv. Increase the counter of the “older” instance by 2 so it becomes the “newer” instance.
5. Go to step 4 for further operations.

Note that this approach does not use more space than a regular fail-safe file. The main difference is that this method implements fail-safe functionality in the host application. Additionally, write operations and flash erases are kept to a minimum.

Figure 4 illustrates the procedure:

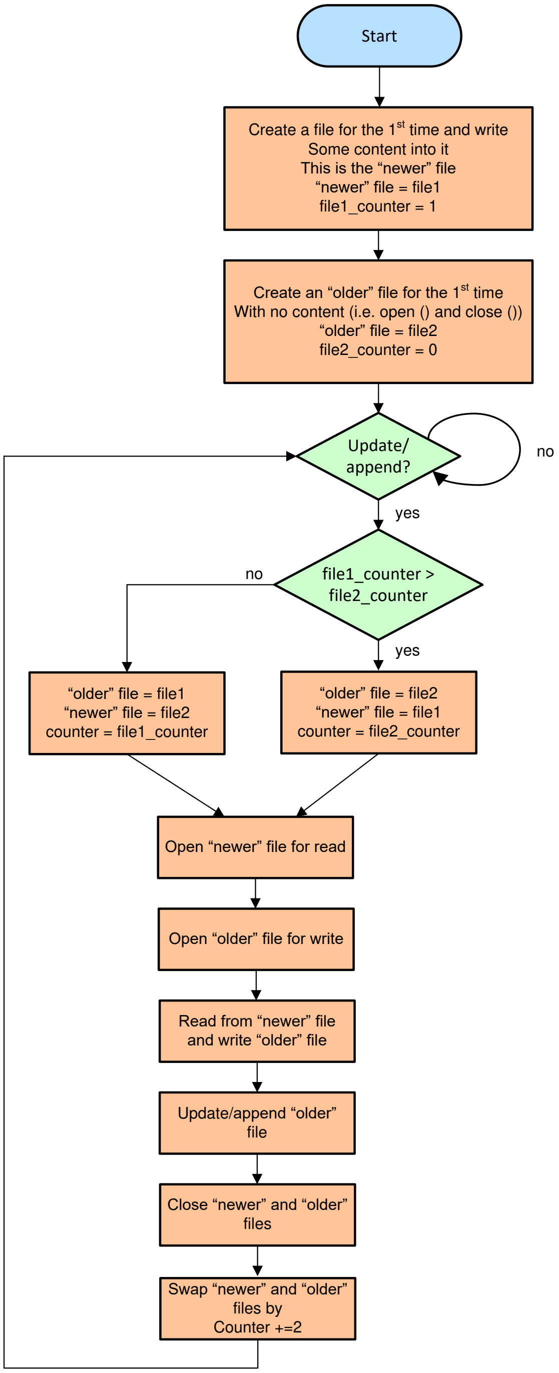
Start
Create a file for the 1st time and write Some content into it This is the “newer” file “newer” file = file1 file1_counter = 1
Create an “older” file for the 1st time With no content (i.e. open () and close ()) “older” file = file2 file2_counter = 0
Update/ append?
no
yes
file1_counter > file2_counter
no
yes
“older” file = file1 “newer” file = file2 counter = file1_counter
“older” file = file2 “newer” file = file1 counter = file2_counter
Open “newer” file for read
Open “older” file for write
Read from “newer” file and write “older” file
Update/append “older” file
Close “newer” and “older” files
Swap “newer” and “older” files by Counter +=2

Figure 4. File Appending Flow

7 Factors to Consider in Designing With Serial Flash

7.1 Serial Flash Vendor and Part Number Selection

Serial flash components from many vendors may appear to be equivalent if only considering memory capacity, however close examination of the serial flash datasheets can reveal significant parametric differences between components in areas such as operating voltage and access times. The serial flash components listed in TI BOM tables for CC3100 and CC3200 reference designs should be used since these serial flash components have been system tested by TI.

Table 8 lists additional parts which were tested with the CC3100R1/CC3200R1 device and found to perform reliably. TI performs a series of system level tests using these parts to ensure robustness under various operating conditions. However, this does not guarantee data integrity under extreme operating conditions as specified in subsequent sections of this document.

Table 8. Serial Flash Parts Tested With the CC3x00 Devices

Vendor	Part Number	Size	Voltage Power	Recommendation
Adesto	AT25SF081	8 Mbit	2.5-3.6 V	Battery and Line powered systems
Adesto	AT25SF161	16 Mbit	2.5-3.6 V	Battery and Line powered systems
Winbond	W25Q80BL	8 Mbit	2.3-3.6 V	Battery and Line powered systems
Spansion	S25FL208K	8 Mbit	2.7-3.6 V	Line Powered systems
Macronix	MX25V8006E	8 Mbit	2.35-3.6 V	Battery and Line powered systems
ISSI	IS25LQ016B	16 Mbit	2.3-3.6 V	Battery and Line powered systems
ISSI	IS25LQ032B	32 Mbit	2.3-3.6 V	Battery and Line powered systems
Micron	N25Q128A	128 Mbit	2.7-3.6 V	Line Powered systems
Shanghai Fudan Micro	FM25Q08	8 Mbit	2.7-3.6 V	Line Powered systems

NOTE: The previously suggested (prior to May 2015) Micron M25PX80 serial flash and other Micron devices in the same series are no longer recommended.

NOTE: The operating voltage of the Wi-Fi subsystem is limited by the operating range of the serial flash. This has to be accounted for by the system designer as part of the overall system design.

7.2 Supported Flash Types

For compatibility with the CC3100/CC3200, the serial flash device must support the following commands and format:

- Uniform sector erase size of 4K
- Command 0x9F (read the device ID [JEDEC]). Procedure: SEND 0x9F, READ 3 bytes.
- Command 0x05 (read the status of the serial flash). Procedure: SEND 0x05, READ 1 byte. Bit 0 is busy and bit 1 is write enable.
- Command 0x06 (set write enable). Procedure: SEND 0x06, read status until write-enable bit is set.
- Command 0xC7 (chip erase). Procedure: SEND 0xC7, read status until busy bit is cleared.
- Command 0x03 (read data). Procedure: SEND 0x03, SEND 24-bit address, read n bytes.
- Command 0x02 (write page). Procedure: SEND 0x02, SEND 24-bit address, write n bytes (0<n<256).
- Command 0x20 (sector erase). Procedure: SEND 0x20, SEND 24-bit address, read status until busy bit is cleared.

7.3 Frequent Write Operations

Serial flash components are subject to a lifetime maximum number of write/erase cycles. This should be accounted for when designing the system application software.

7.3.1 Serial Flash Access by the Wi-Fi System"

A typical serial flash ensures a data endurance of 100K write cycles per sector and 20 years data retention. Table 9 details the maximum number of writes per day to the same sector that will allow the device to operate for a given number of years.

Table 9. Serial Flash Endurance

Desired Product Life in Years	Max Writes/Per Day
20	14
15	18
10	27
5	55

Max number of writes/day = 100000 / (product life in years * 365)

In a CC3100 and CC3200 system, the serial flash can be written by the user application or by the periodic activity of the CC3100/CC3200 on-chip firmware. The total number of writes from these two sources should not exceed the budget for maximum number of flash writes calculated from the desired product lifetime.

7.3.2 Seral Flash Access by the WiFi System

The CC3100/CC3200 uses an external serial flash device, which is required for its basic functionality.

The serial flash content is organized by a file system and every creation or modification of a file translates to operations on the serial flash.

The main files stored in the serial flash are divided into two main groups.

7.3.2.1 System Files

- Service Pack binaries
- System operation: Calibrations, Certifications
- Configurations: WiFi, Device (role), Profiles, Time, and so forth
- Networking: IP, ARP tables

NOTE: The system files are modified by the CC3100/CC3200 upon start-up and on an as-needed basis.

7.3.2.2 User Files

- CC3200 MCU image
- HTML Files - any other user files
- Temporary files – any file been used

NOTE: Note that you can trigger file creations or write operations at unpredictable times.

7.4 Sudden Power Off (power removal during a write/erase phase)

All systems using serial flash are vulnerable to the effects of sudden power removal. As noted in most serial flash data sheets, data corruption may occur if the power is removed while a write/erase operation is in progress. This can happen if the system operating voltage goes below Vmin of the serial flash (2.3 V typical) before the erase/write operation is completed. Figure 5 shows a typical scenario.

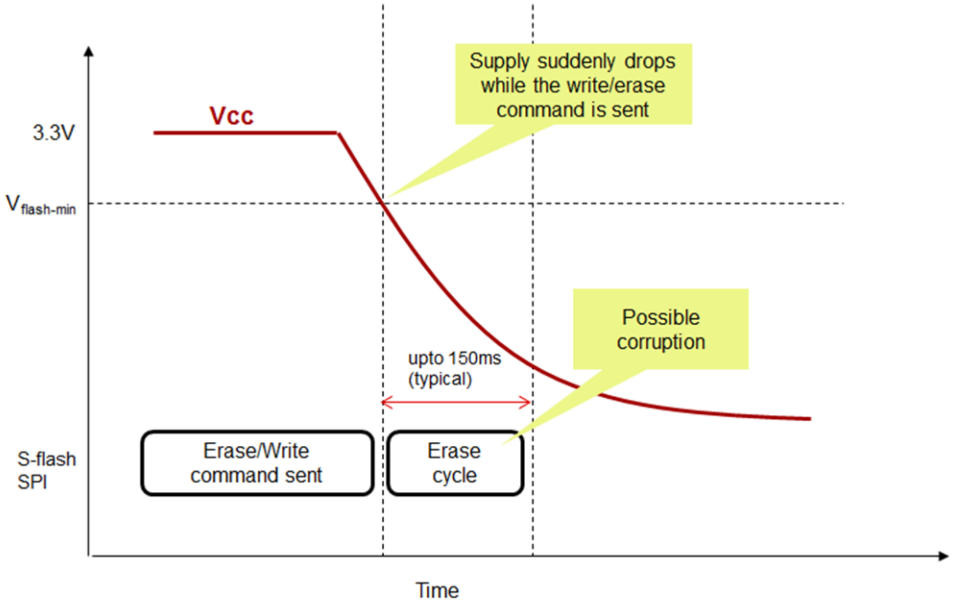

Figure 5. Serial Flash Corruption During a Sudden Power Off Event

This scenario may occur in battery or line powered end equipment.

- Battery Powered Products: Removal of battery from the product without a soft shutdown
- Line Powered Products: Electrical supply failure or sudden unplugging without a soft shutdown

The sections below explain how the potential for serial flash corruption can be minimized in both types of end equipment.

7.4.1 Battery Powered Systems

In a battery powered system, there is a chance that sudden removal of power could coincide with an ongoing serial flash access. The chance of this happening can be minimized by making it difficult to remove the batteries while the product is in use, for example, in a Wi-Fi weighing scale, where you stand on the scale when it is in use. Alternatively, product instructions could discourage the removal of the batteries while the product is in use. In systems desiring a higher degree of protection, a soft power down push-button could also be provided that maps to a GPIO of the system processor, giving it advanced warning that you want to remove the battery. In such a system the processor could use an LED to indicate that soft shut down has completed and that the batteries can be removed.

7.4.2 Line Powered Systems

In an AC line powered system, the failure of the grid can cause the supply of the Wi-Fi subsystem to drop suddenly. In the unlikely event this coincides with the erase operation of the serial flash, there is a chance of data corruption. One of the ways to minimize the chance of a flash corruption would be to ensure that the DC voltage ramps down slowly after the input power is removed. This can be achieved with the help of bulk capacitors which hold the charge while input supply is removed. The embedded system would sense the sudden fall in the input voltage and then initiate a soft shutdown of the Wi-Fi sub-system thereby safely completing all serial flash operations before Vflash-min is reached. The charge stored on the capacitor would be used during this brief interval. Such a sequence is illustrated in Figure 6.

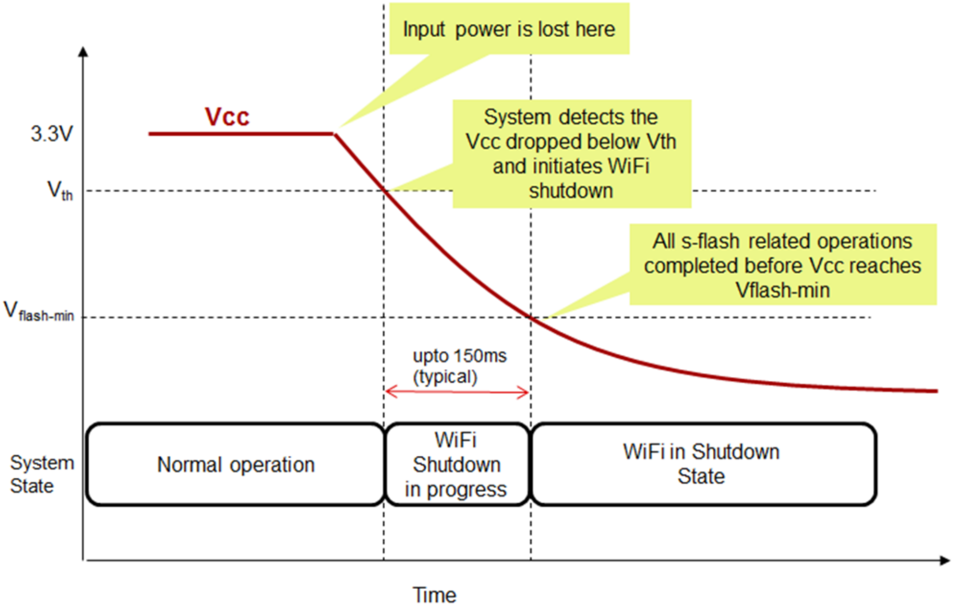
Input power is lost here
Vcc
3.3V
V_{th}
System detects the Vcc dropped below Vth and initiates WiFi shutdown
All s-flash related operations completed before Vcc reaches Vflash-min
$V_{flash-min}$
upto 150ms (typical)
System State
Normal operation
WiFi Shutdown in progress
WiFi in Shutdown State
Time

Figure 6. Serial Flash Corruption Avoided Through Use of a Bulk Capacitor

8 Design Recommendations for Ensuring the Integrity of the Power Supply to the Serial Flash

8.1 *Overview*

The CC3100 and CC3200 are proven, robust WLAN solutions when operated in accordance with the supply and signal parameters described in the data sheet with the latest service pack. This document describes design techniques to maximize system robustness including minimizing the possibility of inadvertent corruption of the serial Flash memory connected to CC3100 or CC3200 for battery powered and hybrid line/battery powered designs. The techniques described focus on ensuring the integrity of the power supply to the serial flash to avoid situations where the supply voltage may fall below the minimum threshold specified by the serial flash manufacturer, which may cause corruption of flash data during write or erase operations. Depending on the nature of such corruption systems may or may not continue to operate, and in some cases physical access to the system may be required in order to recover. This section will cover design techniques to maximize system robustness in battery powered and hybrid line/battery power systems.

8.2 Key Points

- This document applies primarily to systems that are:
  - Powered from batteries
  - Powered by a hybrid line power and battery power scheme
  - This includes designs where the CC3x00 connects directly to the battery and systems where a DC2DC converter is used between the battery and the CC3x00
  - If the application uses a DC2DC converter then designers should ensure the output of the DC2DC converter satisfies the supply requirements of both the CC3x00 and the attached serial flash device.
- CC3x00 devices should be enabled only when supply voltage is greater than or equal to 2.3 V. This minimum is typically determined by the serial flash minimum supply voltage and not the CC3x00 minimum supply voltage which is defined in the data sheet as 2.1 V. The key point is that the supply must tolerate a CC3x00 transmit current or calibration current load, as specified in the data sheet, without drooping below 2.3 V so the unloaded supply voltage may have to be greater than 2.3 V to account for internal resistance effects of the supply.
- The supply voltage applied to the CC3x00 should never exceed 3.8V, specified as the absolute maximum supply voltage in the data sheet. And corresponding absolute maximum voltage constraints from the chosen serial flash data sheet must also be followed.
- Follow TI serial flash design guidelines for CC3x00
- Minimize the number of application writes to flash, especially after reset. For example make sure that application configuration writes to flash happen only at initial reset and not every time the CC3x00 is reset.
- For maximum system robustness consider using a serial flash like the Macronix MX25R6435.This device supports a wide supply voltage range which tends to improve system immunity to supply fluctuations.
- Keep in mind that CC3x00 WLAN transmission can result in sudden increases in the loading on the power supply and this may result in a momentary decrease in supply voltage. For a description on how to handle supply brown out, see the *Brown-Out and Black-Out* section of the *CC3100 SimpleLink™ Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications Data Sheet* and the *CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU Data Sheet*. Specific design scenarios are described in Table 10.

Table 10. Design Scenarios

Scenario	Recommended Approach
All MCU hosted CC3100 systems	Host MCU should own the task of monitoring VBAT to ensure it never drops below 2.3 V even under full load conditions.
Legacy Self-hosted, standalone CC3200	For legacy systems using SDKs prior to Service pack 1.0.1.6 and SDK 1.2.0 the CC3200 initialization sequence will turn on the internal network processor and may initiate Wi-Fi activity prior to application execution on the MCU. This can cause a sudden increase in the power supply loading. In these systems if there is a possibility that such loading can cause the supply voltage to sag below 2.3 V then the supply voltage should be monitored by a device external to CC3200.
New Self-hosted, standalone CC3200	For new systems using Service Pack 1.0.1.6 and SDK 1.2.0 or later then the CC3200 initialization will not turn on the internal networking processor function so the CC3200 application can start running prior to Wi-Fi activity. In such a scenario the application code can use the CC3200 ADC to monitor battery voltage but keep in mind that the ADC accuracy needs to be factored into any decision to enable or disable the NWP for normal operation. For more details on optimal detection and handling of consecutive resets owing to supply brownout to maximize system robustness see the last section of this document.

Other comments on circuit and software design: For systems requiring an external DC2DC power converter the TI TPS61029 may be worth considering.

8.3 Brown-Out Mitigation Techniques for New Self-Hosted CC3200 Designs

This section describes how to mitigate the effects of a substantially discharged battery on a self-hosted CC3200 system. It is assumed that the CC3200 device is the main controller and has the ability to control all high power components in the system and that Service Pack [1.0.1.6] and SDK [1.2.0] or later have been installed.

The problem: The brownout problem can occur when operating with a substantially discharged battery which has enough energy to power up the CC3200 processor, but not enough to power the Wi-Fi transmitter. This can cause a loop where the CC3200 device powers up, reaches the point where it does some high power activity, this activity causes the battery voltage to drop below brown out threshold causing a reset. Once reset, the device consumes no power, the voltage rises back above brown out threshold and the device powers up again.

The solution: The suggested approach to avoid the above cycle is to use the secondary bootloader (like the one used in over-the-air updates) to load the user application and keep track of whether the application was loaded successfully (without causing another brown out event). This is done by keeping a counter in an on chip register (OCR) which will most likely be retained if voltage drops due to excessive power usage. The power up flow should be as described in the scheme below.

9 Recommended Best Practices

The following list of best design practices can help improve the overall reliability of the serial flash by minimizing frequent writes:

- Avoid frequent usage of API functions that write to the serial flash
- The creation and deletion of files also require updates to the FAT table – as a good practice avoid deleting files when not required, rewrite can be done over existing file. This will reduce unnecessary access to the FAT table.
- Avoid accessing the serial flash when the voltage is below the minimum serial flash operation voltage.
- Make sure you choose your serial flash based on the system you are developing – for example when using a serial flash rated for 2.7 V you must ensure that supply remains above 2.7 V.
- Adding an indication before power off (graceful shutdown) and making a call to sl_Stop() in the program is always the best approach

10 Implications of Data Integrity Compromise to CC3100/CC3200

The CC3200 system may become inoperable if the MCU image is corrupted. In the case of the CC3100 or CC3200, where the FAT or service pack are corrupted, the device may start up with a ROM image instead of the correct service pack. There is also a chance that the device would lock-up without any valid response.

10.1 Recovery

To recover from serial flash corruption the serial flash has to be reformatted and reprogrammed using the universal asynchronous receiver/transmitter (UART) interface, or directly using a SPI flash programmer.

11 References

- Texas Instruments: *CC3100 SimpleLink™ Wi-Fi® Network Processor, Internet-of-Things Solution for MCU Applications Data Sheet*
- Texas Instruments: *CC3200 SimpleLink™ Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU Data Sheet*
- Texas Instruments: SimpleLink Wi-Fi CC3100, CC3200 UniFlash User's Guide (SWRU558)

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265