AN13542

Porting Tiny Flash Database Stack on LPC5500 Series

Rev. 0 — 10 February 2022 Application Note

Contents

1 Introduction

When migrating from LPC54000 to LPC5500 series, one of the biggest obstacles is the lack of real EEPROM memory in LPC5500 series. Several software emulation layers exist to address this issue. However, while providing a seamless emulation of the EEPROM interface, they all have significant drawbacks. For example, the wear on the underlying Flash memory is excessive and data may lose if the power interrupts while the write is ongoing. This document introduces a better approach. This approach upgrades Flash EEPROM concept and makes Flash into a tiny database. Use a simple Key-Value (KV) Paris to read and write user data (parameters). At the same time, the software considers the wear leveling and power loss protection.

This open source software requires minor changes to the application flow. It yields a robust application parameter management solution with minimal wear of the MCU Flash memory.

1.1 LPC5500 Flash performance

The on-chip flash of LPC5500 series contains up to 640 kB on-chip flash program memory with flash accelerator and 512 byte page for erasing and writing.

Figure 1 shows some key Flash parameters.

Table 23. Flash characteristics[2]

T_{amb} = –40 °C to +105 °C, unless otherwise specified.

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
N_{endu}	endurance	Page erase/program, T_{amb} = –40 °C to +85 °C	[1]	100000	-	-	cycles
		Mass erase/program, T_{amb} = –40 °C to +85 °C		100000	-	-	cycles
		Page erase/program T_{amb} = –40 °C to +105 °C,		10000	-	-	cycles
		Mass erase/program T_{amb} = –40 °C to +105 °C,		10000	-	-	cycles
t_{ret}	retention time	< 1k erase/program cycles		25	-	-	years
		≥ 1k erase/program cycles		15	-	-	years
t_{er}	erase time	1 page or multiple pages		-	2.0	-	ms
t_{prog}	programming time			-	1.09	-	ms

[1] Number of erase/program cycles.

[2] Flash operations (erase, blank check, program) and reading single word can only be performed for CPU frequencies of up to 100 MHz. Cannot be performed for frequencies above 100 MHz.

Figure 1. LPC5500 flash parameters

As shown in Figure 1, compared to LPC54000 series, the page erase and page programming time are very fast.

1.1.1 Flash API in SDK

The Flash operation is via In Application Programming (IAP) API which is defined in MCUXpresso SDK. The SDK package can be download from MCUXpresso SDK Builder.

For detailed Flash API description, see **Chapter 9 FLASH APIs** in *LPC55S6x/LPC55S2x/LPC552x User manual* (document UM11126).

SDK provides examples of how to use flash API to operate Flash.

Flash example is in the SDK `driver_example` folder:

\SDK_2.X.X_LPCXpresso55S69\boards\lpcxpresso55s69\driver_examples\flashiap

If you are not familiar with LPC5500 series flash operation, go to **Chapter 9 FLASH APIs** and the `flashiap` example of SDK. Also, SDK provides a unified API across all LPC5500 series. When migrating from one chip to another inside LPC5500 series, there is no extra work. This application note uses LPC55S69 as an example and experiment platform.

2 Porting FlashDB on LPC5500 series

2.1 FlashDB introduction

FlashDB is an ultra-lightweight embedded database. It provides data storage solutions for embedded products. Different from traditional database based on file system, FlashDB combines the features of Flash. It has strong performance and reliability. Under the premise of ensuring low resource occupation, extend the service life of Flash as much as possible.

Home page: https://github.com/armink/FlashDB

Documentation: https://armink.github.io/FlashDB/#/

The FlashDB supports two database modes:

- Key-Value Database (KVDB): It is a non-relational database that stores data as a collection of key-value pairs. In KVDB, the key is used as a unique identifier. KVDB has simple operations and strong scalability.
- Time Series Database (TSDB): It stores data in time sequence. TSDB data has a timestamp, a large amount of data storage, and high insertion and query performance.

2.2 Environment setup

2.2.1 Hardware setup

Hardware: LPCXpresso55S69EVK.

Make sure you are familiar with this board. Find the getting stared tutorial from:

https://www.nxp.com/document/guide/get-started-with-the-lpc55s69-evk:GS-LPC55S69-EVK

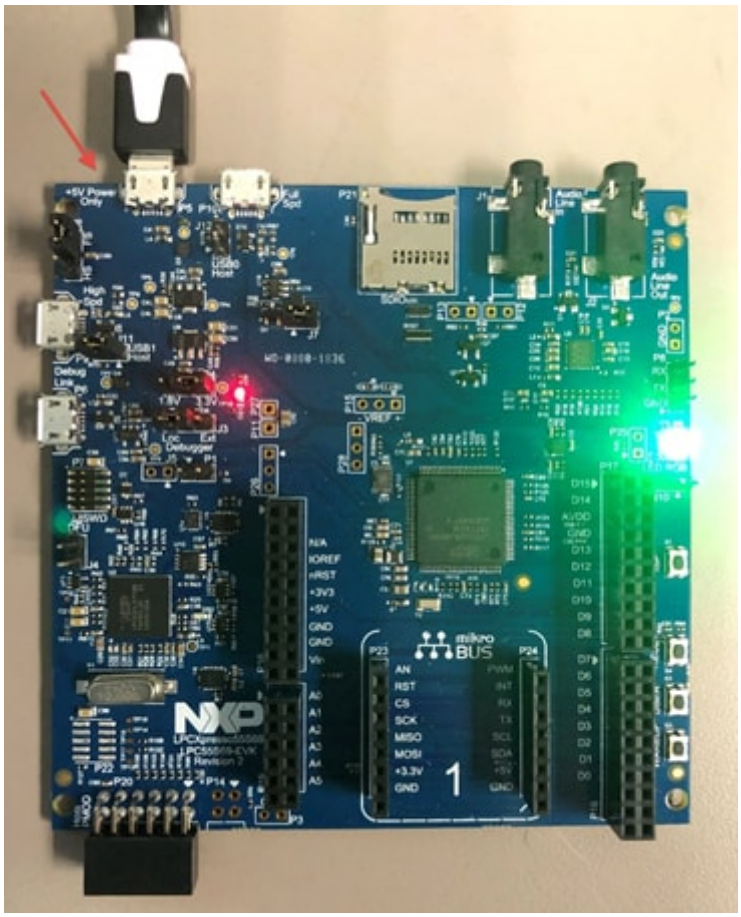

Figure 2. LPC55S69 EVK

2.2.2 Software setup

Download FlashDB source from the github page. The folder structure of FlashDB is simple, as shown in Figure 3.

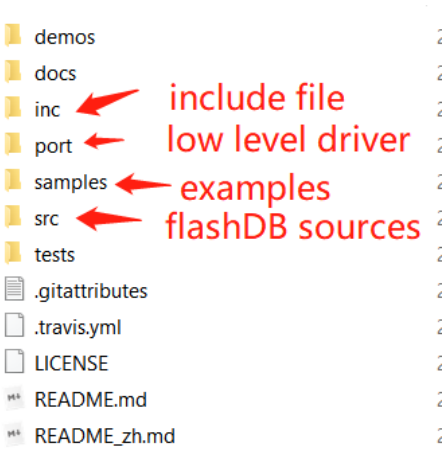

Figure 3. FlashDB folder structure

In the IDE setting, add the `inc` folder in the include path setting and add `src`, `samples`, and `port` into project, as shown in Figure 4.

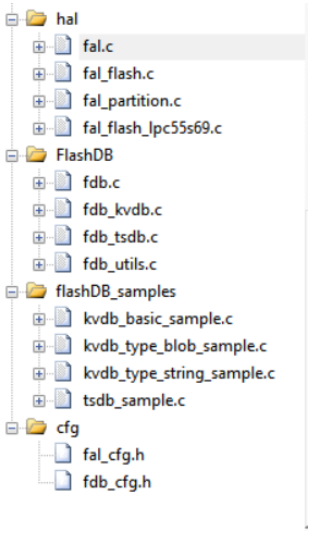

Figure 4. Adding FlashDB source file to your project

NOTE

`fal_flash_lpc55s69.c` must be created by users.

2.3 Implementing flash driver interface

The most important step for porting FlashDB stack is to implement low-level Flash operation API. In FlashDB, `fal`, a simple flash abstraction layer, manages the low-level Flash operation. To define specific flash device objects, implement the operation functions of **init**, **read**, **write**, and **erase** according to their own Flash conditions.

2.3.1 Init

```
static int init(void)
```

To define a global Flash configuration instance and initiate LPC on-chip Flash, call `FLASH_Init`.

```
static flash_config_t flashInstance;

static int init(void)
{
    FLASH_Init(&flashInstance);
    return 0;
}
```

Figure 5. Init

2.3.2 Read

```
static int read(long offset, uint8_t *buf, size_t size)
```

```
static int read(long offset, uint8_t *buf, size_t size)
{
    uint32_t status;
    status = FLASH_Read(&flashInstance, lpc_onchip_flash.addr + offset, buf, size);
    return (status == kStatus_Success)?(size):(0);
}
```

Figure 6. Read

NOTE

To read Flash data, use FLASH_Read API not AHB reading. Thus, when AHB reads an erased (empty) page of Flash, the Hardfault issue is prevented.

2.3.3 Erase

```
static int write(long offset, const uint8_t *buf, size_t size)
```

```
static int erase(long offset, size_t size)
{
    uint32_t status;
    uint32_t addr = lpc_onchip_flash.addr + offset;
    uint32_t pz_ali_addr = ALIGN_DOWN(addr, FLASH_PAGE_SIZE);

    status = FLASH_Erase(&flashInstance, pz_ali_addr, ALIGN_UP(size, FLASH_PAGE_SIZE), kFLASH_ApiEraseKey);
    return (status == kStatus_Success)?(size):(0);
}
```

Figure 7. Erase

2.3.4 Write

```
static int erase(long offset, size_t size)
```

```
static int write(long offset, const uint8_t *buf, size_t size)
{
    int i;
    uint32_t status;
    uint32_t addr = lpc_onchip_flash.addr + offset;
    uint32_t pz_ali_addr = ALIGN_DOWN(addr, FLASH_PAGE_SIZE);
    uint32_t offset_in_sec = addr - pz_ali_addr;


    status = FLASH_VerifyErase(&flashInstance, pz_ali_addr, ALIGN_UP(size, FLASH_PAGE_SIZE));
    if (status == kStatus_Success)
    {
        memset(write_cache, 0xFF, sizeof(write_cache));
    }
    else
    {
        read(pz_ali_addr, write_cache, sizeof(write_cache));
        erase(pz_ali_addr, size);
        for(i=0; i< size; i++)
        {
            if(write_cache[offset_in_sec+i] != 0xFF)
            {
                break;
            }
        }
    }

    memcpy(&write_cache[offset_in_sec], buf, size);
    FLASH_Program(&flashInstance, pz_ali_addr, write_cache, sizeof(write_cache));

    return size;
}
```

Figure 8. Write

The minimum program/erase unit of LPC5500 series is one page. It equals to 512 bytes. Before programming, make sure that the page is erased. We use `FLASH_VarifyErase` to verify whether the current page is an erased page. If the current page is not an erased page, perform a read-modify-write operation on that page.

2.4 Testing

FlashDB provide simple example test code to demonstrate basic usage of flashDB.

Add test code in `main.c`:

```
static uint32_t boot_count = 0;
static time_t boot_time[10] = {0, 1, 2, 3};
/* default KV nodes */
static struct fdb_default_kv_node default_kv_table[] = {
        {"username", "armink", 0}, /* string KV */
        {"password", "123456", 0}, /* string KV */
        {"boot_count", &boot_count, sizeof(boot_count)}, /* int type KV */
        {"boot_time", &boot_time, sizeof(boot_time)},    /* int array type KV */
};
/* KVDB object */
static struct fdb_kvdb kvdb = { 0 };
static int counts = 0;
```

Figure 9. Defining FlashDB instance and test KV data

In `main.c`, initialize FlashDB instance and test FlashDB function with `kv` example.

```
{ /* KVDB Sample */
    struct fdb_default_kv default_kv;

    default_kv.kvs = default_kv_table;
    default_kv.num = sizeof(default_kv_table) / sizeof(default_kv_table[0]);
    /* set the lock and unlock function if you want */
    fdb_kvdb_control(&kvdb, FDB_KVDB_CTRL_SET_LOCK, (void *)lock);
    fdb_kvdb_control(&kvdb, FDB_KVDB_CTRL_SET_UNLOCK, (void *)unlock);
    /* Key-Value database initialization
     *
     *       &kvdb: database object
     *       "env": database name
     * "fdb_kvdb1": The flash partition name base on FAL. Please make sure it's in FAL partition table.
     *              Please change to YOUR partition name.
     * &default_kv: The default KV nodes. It will auto add to KVDB when first initialize successfully.
     *        NULL: The user data if you need, now is empty.
     */
    result = fdb_kvdb_init(&kvdb, "env", "fdb_kvdb1", &default_kv, NULL);

    if (result != FDB_NO_ERR) {

        return -1;
    }

    /* run basic KV samples */
    kvdb_basic_sample(&kvdb);
      /* run string KV samples */
    kvdb_type_string_sample(&kvdb);
      /* run blob KV samples */
    kvdb_type_blob_sample(&kvdb);
}
```

Figure 10. Testing FlashDB using kvdb_basic sample

The code can be found in the **\FlashDB\demos** folder. After adding the code, build/compile project and download to MCU. Open UART terminal software and reset the board. There is log output.

When the board is reset, the variable, `boot_count`, increase every time. The reason is that the example code read `boot_cnt` variable from database, increase by one, and save to database, as shown in Figure 11.

```
[D/FAL] (fal_flash_init:65) Flash device |              lpc_onchip | addr: 0x00000000 | len: 0x00080000 | blk_size:
0x00000200 |initialized finish.
□[32;22m[I/FAL] ================== FAL partition table ==================□[0m
□[32;22m[I/FAL] | name      | flash_dev |   offset   |    length  |□[0m
□[32;22m[I/FAL] ------------------------------------------------------□[0m
□[32;22m[I/FAL] | fdb_tsdb1 | lpc_onchip | 0x0001a000 | 0x00002000 |□[0m
□[32;22m[I/FAL] | fdb_kvdb1 | lpc_onchip | 0x0001c000 | 0x00004000 |□[0m
□[32;22m[I/FAL] ======================================================□[0m
□[32;22m[I/FAL] Flash Abstraction Layer (V0.5.0) initialize success.□[0m
[FlashDB][kv][env] (../FlashDB/src/fdb_kvdb.c:1608) KVDB size is 16384 bytes.
[FlashDB] FlashDB V1.0.99 is initialize success.
[FlashDB] You can get the latest version on https://github.com/armink/FlashDB .
[FlashDB][sample][kvdb][basic] ================== kvdb_basic_sample ==================
[FlashDB][sample][kvdb][basic] get the 'boot_count' value is 32
[FlashDB][sample][kvdb][basic] set the 'boot_count' value to 33
[FlashDB][sample][kvdb][basic] ======================================================
[FlashDB][sample][kvdb][string] ================== kvdb_type_string_sample ==================
[FlashDB][sample][kvdb][string] create the 'temp' string KV, value is: 36C
[FlashDB][sample][kvdb][string] get the 'temp' value is: 36C
[FlashDB][sample][kvdb][string] set 'temp' value to 38C
[FlashDB][sample][kvdb][string] delete the 'temp' finish
[FlashDB][sample][kvdb][string] ============================================================
[FlashDB][sample][kvdb][blob] ================== kvdb_type_blob_sample ==================
[FlashDB][sample][kvdb][blob] create the 'temp' blob KV, value is: 36
[FlashDB][sample][kvdb][blob] get the 'temp' value is: 36
[FlashDB][sample][kvdb][blob] set 'temp' value to 38
[FlashDB][sample][kvdb][blob] delete the 'temp' finish
[FlashDB][sample][kvdb][blob] ============================================================
Done!
```

Figure 11. FlashDB example log

3 Summary

This application note summarizes Flash performance and key parameter of LPC5500 series. It describes how to port the FlashDB stack on to LPC5500 series.

4 Reference

1. https://github.com/armink/FlashDB
2. https://armink.github.io/FlashDB/#/

5 Revision history

Rev.	Date	Description
0	10 February 2022	Initial release

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

Right to make changes - NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP, the NXP logo, NXP SECURE CONNECTIONS FOR A SMARTER WORLD, COOLFLUX,EMBRACE, GREENCHIP, HITAG, ICODE, JCOP, LIFE, VIBES, MIFARE, MIFARE CLASSIC, MIFARE DESFire, MIFARE PLUS, MIFARE FLEX, MANTIS, MIFARE ULTRALIGHT, MIFARE4MOBILE, MIGLO, NTAG, ROADLINK, SMARTLX, SMARTMX, STARPLUG, TOPFET, TRENCHMOS, UCODE, Freescale, the Freescale logo, AltiVec, CodeWarrior, ColdFire, ColdFire+, the Energy Efficient Solutions logo, Kinetis, Layerscape, MagniV, mobileGT, PEG, PowerQUICC, Processor Expert, QorIQ, QorIQ Qonverge, SafeAssure, the SafeAssure logo, StarCore, Symphony, VortiQa, Vybrid, Airfast, BeeKit, BeeStack, CoreNet, Flexis, MXC, Platform in a Package, QUICC Engine, Tower, TurboLink, EdgeScale, EdgeLock, eIQ, and Immersive3D are trademarks of NXP B.V. All other product or service names are the property of their respective owners. AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, µVision, Versatile are trademarks or registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved. Oracle and Java are registered trademarks of Oracle and/or its affiliates. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org. M, M Mobileye and other Mobileye trademarks or logos appearing herein are trademarks of Mobileye Vision Technologies Ltd. in the United States, the EU and/or other jurisdictions.

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 10 February 2022
Document identifier: AN13542