AN13165
LPC546xx SPIFI Usage and Performance

Rev. 0 — February 25, 2021
Application Note

Contents

1 Introduction

The LPC54628 is a family of Arm® Cortex®-M4 based microcontrollers used in embedded applications. The SPI flash interface is available on all LPC546xx devices. Compared to parallel flash devices with higher pin count, the SPI Flash Interface (SPIFI) allows low-cost serial flash memories to the CPU with little performance penalty.

This application note describes how to implement Execute In Place (XIP) feature with SPIFI, data access and performance benchmark with KEIL.

2 SPI Flash Interface (SPIFI)

2.1 Features

- Quad SPIFI to external flash
- Transfer rate up to SPIFI_CLK/2 bytes per second
- Code in the serial flash memory can be executed same as in the CPU's internal memory directly into the CPU memory space
- Supporting 1-, 2-, and 4-bit bidirectional serial protocols
- Half-duplex protocol compatible with various vendors and devices
- Maximum supported bit rate for SPIFI mode: 100 Mbit/s

2.2 SPIFI operation modes

SPIFI has two operational modes:

- Memory mode - whereby the contents of FLASH are memory mapped in the chip
- Command mode - Whereby the user can manually construct command sequences for the flash

3 XIP feature implement

3.1 SPIFI initialization

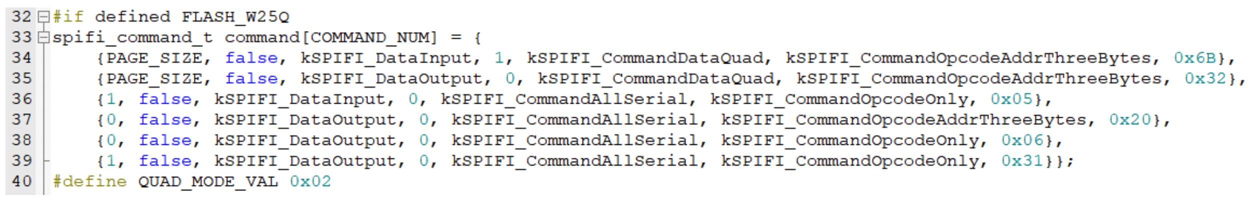
SPIFI is not initialized when the system is booting and it must be initialized before running code which is allocated in SPIFI Flash. Therefore, before the SPIFI initialization, allocate all related codes to run in RAM or internal FLASH. Users can change code allocation by modifying the linker file.

The demo project, loaded at *SDK_2.8.2_LPCXpresso54628_SPIFI\boards\lpcxpresso54628\demo_apps\spifi_usage\MDK*, shows how to enable SPIFI and test the XIP feature.

1. Initialize pins for SPIFI.

In the `BOARD_InitPins()`, we initialize GPIO for SPIFI. Initialize `P0_23`, `P0_24`, `P0_25`, `P0_26`, `P0_27`, `P0_28` to SPIFI CS, `IO(0)`, `IO(1)`, CLK, `IO(3)`, `IO(4)`.

2. Complete the command table for SPIFI flash.



```
#if defined FLASH_W25Q
spifi_command_t command[COMMAND_NUM] = {
    {PAGE_SIZE, false, kSPIFI_DataInput, 1, kSPIFI_CommandDataQuad, kSPIFI_CommandOpcodeAddrThreeBytes, 0x6B},
    {PAGE_SIZE, false, kSPIFI_DataOutput, 0, kSPIFI_CommandDataQuad, kSPIFI_CommandOpcodeAddrThreeBytes, 0x32},
    {1, false, kSPIFI_DataInput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x05},
    {0, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeAddrThreeBytes, 0x20},
    {0, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x06},
    {1, false, kSPIFI_DataOutput, 0, kSPIFI_CommandAllSerial, kSPIFI_CommandOpcodeOnly, 0x31}};
#define QUAD_MODE_VAL 0x02
```

Figure 1. SPIFI flash command table

In the `spifi_flash.c`, define macro `FLASH_W25Q` to select the correct flash operation command table for SPIFI flash. Users can refer to the specifications of SPIFI Flash to complete the command table following the format of `spifi_command_t struct`.

3. Implement the initialization function for SPIFI.

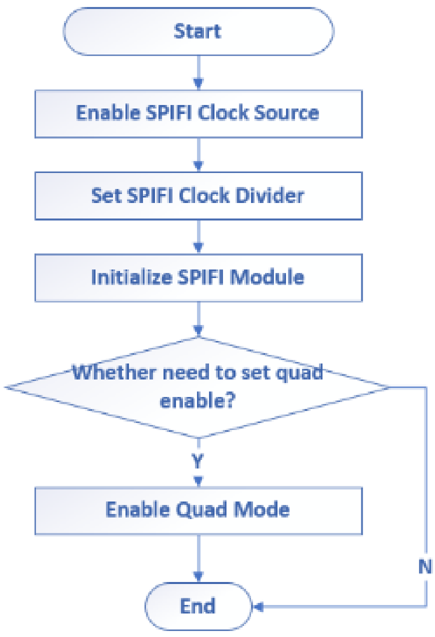
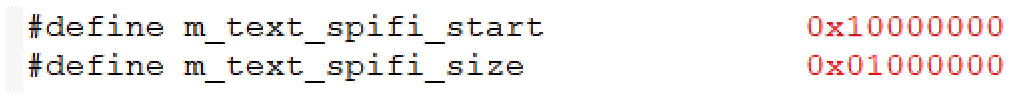


Figure 2. SPIFI initialization flowchart

The `spifi_flash_init()` function in `spifi_flash.c` will initialize the SPIFI and set the SPIFI flash to work with quad mode if needed.

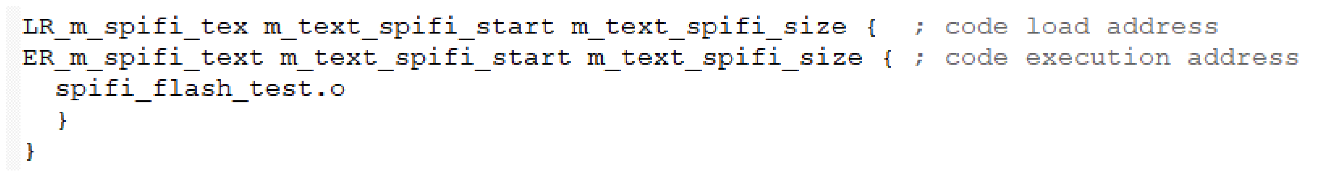
3.2 Code location edit

To allocate SPIFI flash initialization code to internal flash and some demo code into SPIFI flash, we can edit linker file to accomplish it. In KEIL IDE, we can modify the *LPC54628J512_flash.scf* file as below.



```
#define m_text_spifi_start              0x10000000
#define m_text_spifi_size               0x01000000
```

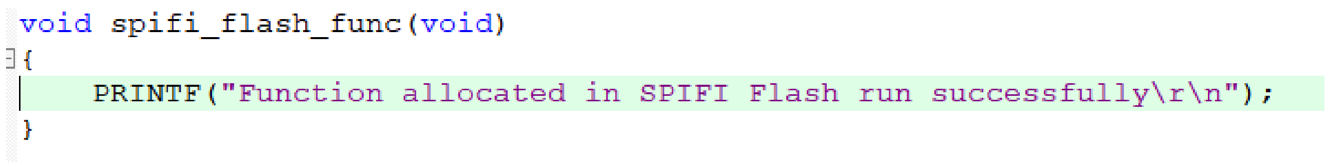
Figure 3. SPIFI Flash start address and size definition



```
LR_m_spifi_tex m_text_spifi_start m_text_spifi_size {  ; code load address
ER_m_spifi_text m_text_spifi_start m_text_spifi_size { ; code execution address
  spifi_flash_test.o
  }
}
```

Figure 4. Code load and execution address

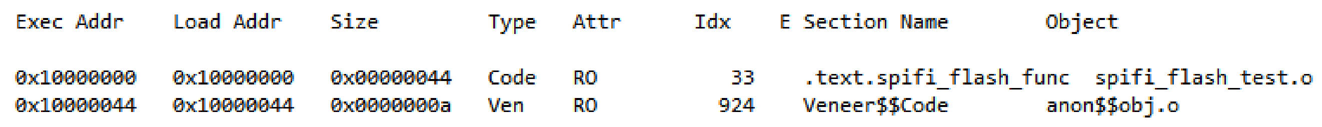
In the demo project, we implement a function named `spifi_flash_func()` in *spifi_flash_test.c* to print a string by debug uart.



```
void spifi_flash_func(void)
{
    PRINTF("Function allocated in SPIFI Flash run successfully\r\n");
}
```

Figure 5. Test function

After compiling the project and checking the map file, we can find that the `spifi_flash_func()` function has been allocated to SPIFI flash.



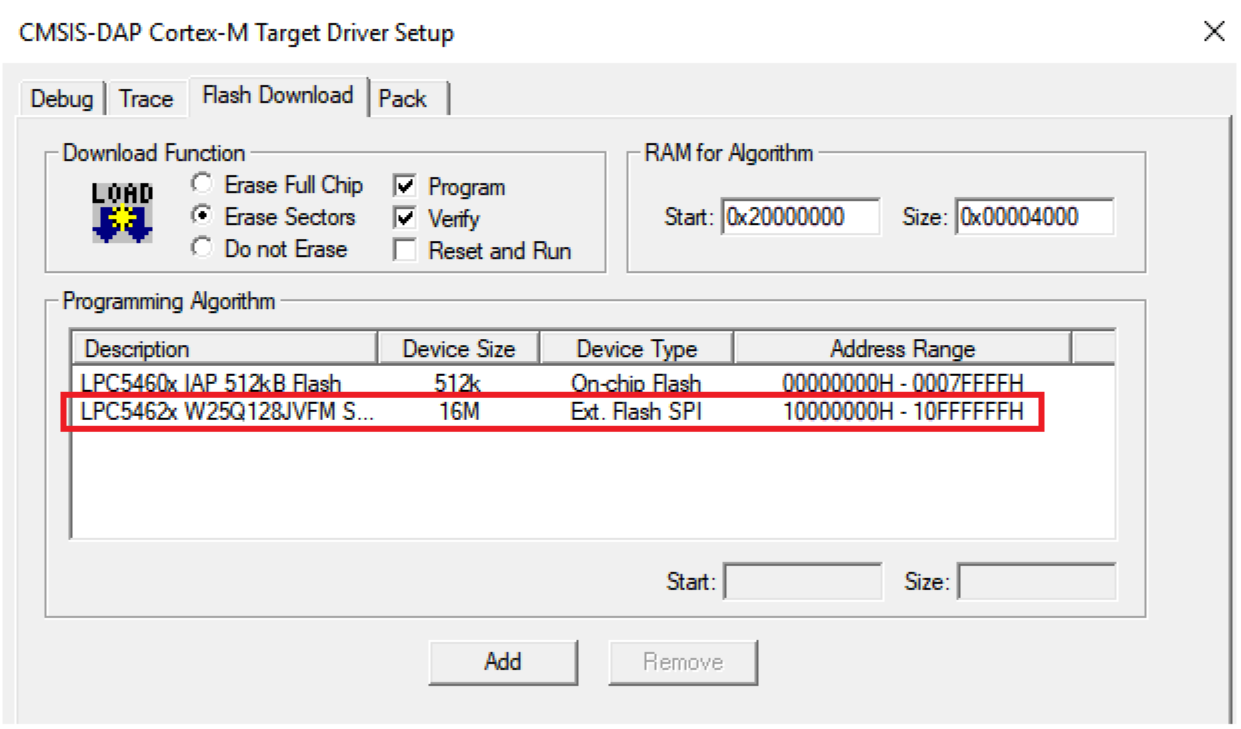
```
Exec Addr    Load Addr    Size         Type   Attr      Idx    E Section Name        Object

0x10000000   0x10000000   0x00000044   Code   RO          33     .text.spifi_flash_func  spifi_flash_test.o
0x10000044   0x10000044   0x0000000a   Ven    RO         924     Veneer$$Code           anon$$obj.o
```

Figure 6. Map file

3.3 Code download

To download codes to SPIFI flash, add external flash programming algorithm in the IDE. At LPC54628 EVK, the external flash connected to SPIFI is W25Q128JVFM, so add the corresponding programming algorithm.



Download Function: ( ) Erase Full Chip [✓ Erase Sectors] ( ) Do not Erase; [✓] Program [✓] Verify [ ] Reset and Run

RAM for Algorithm: Start: 0x20000000 Size: 0x00004000

Programming Algorithm

Description	Device Size	Device Type	Address Range
LPC5460x IAP 512kB Flash	512k	On-chip Flash	00000000H - 0007FFFFH
LPC5462x W25Q128JVFM S...	16M	Ext. Flash SPI	10000000H - 10FFFFFFH

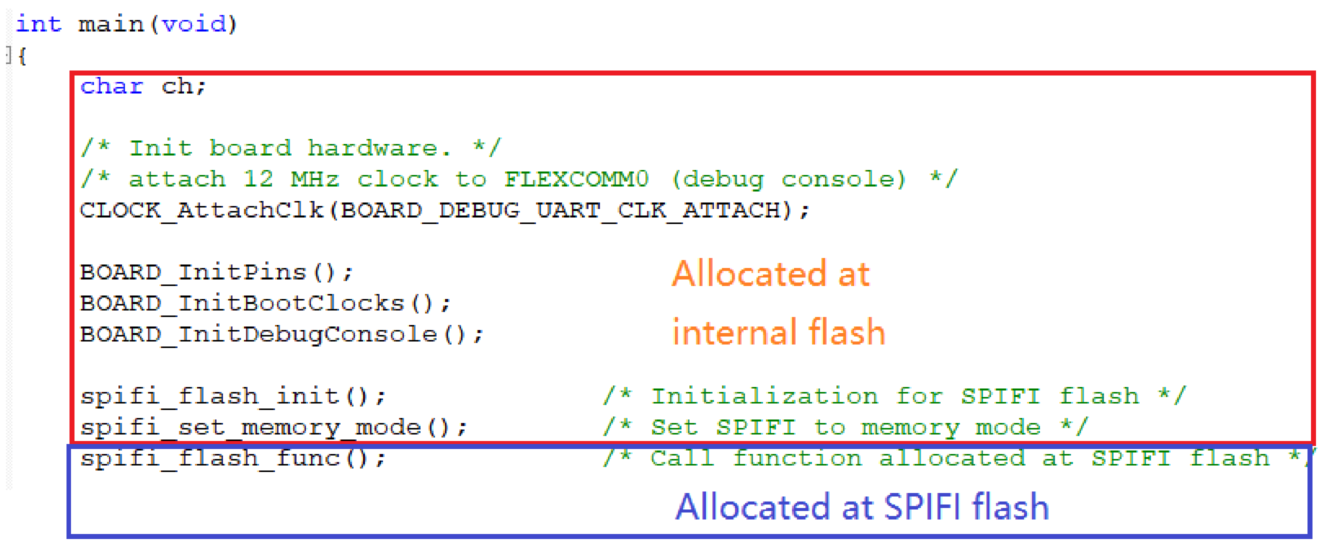
Start: Size:

Figure 7. Programming algorithm configurations

Please note the size of **RAM for Algorithm**. Sometimes we shall extend the size value big enough to load programming algorithm code.

3.4 Running the demo code

In the main function, we call the function allocated to SPIFI flash and check the print output.



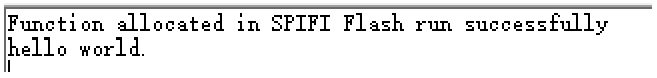
```
int main(void)
{
    char ch;

    /* Init board hardware. */
    /* attach 12 MHz clock to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    spifi_flash_init();              /* Initialization for SPIFI flash */
    spifi_set_memory_mode();         /* Set SPIFI to memory mode */
    spifi_flash_func();              /* Call function allocated at SPIFI flash */
```

Figure 8. Main function



```
Function allocated in SPIFI Flash run successfully
hello world.
```

Figure 9. Print output

As shown in Figure 9, the function allocated to SPIFI flash can run successfully. It means that the XIP feature is successfully enabled.

4 SPIFI Flash data access

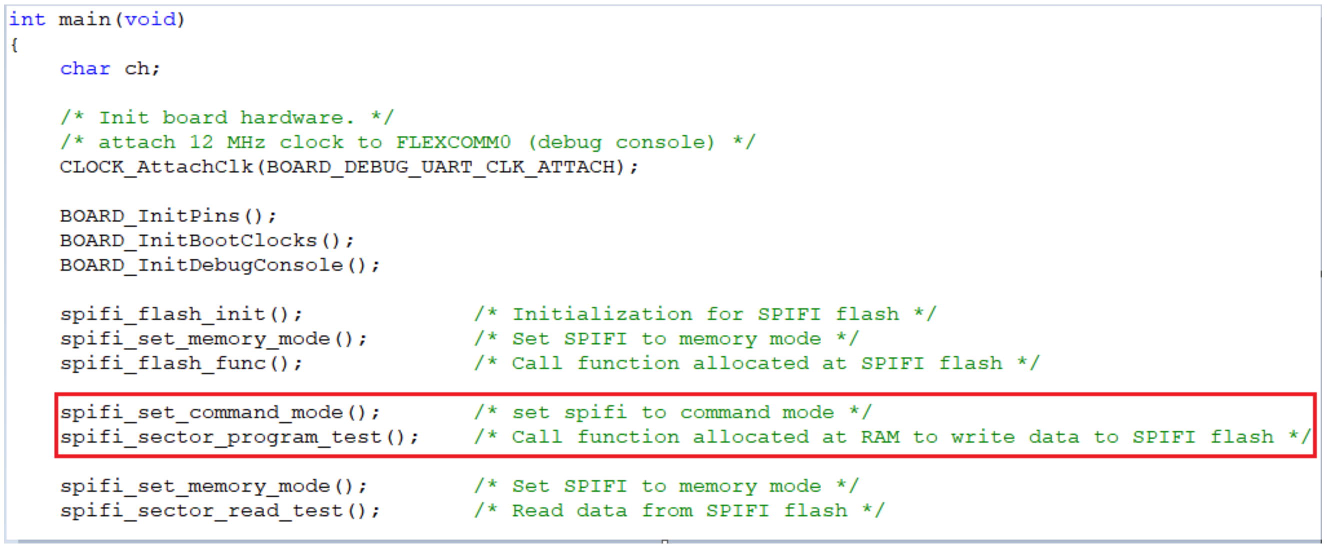
4.1 Write data to SPIFI Flash

To write data to SPIFI flash when code is running, follow the two conditions as below:

- The code related to write data to SPIFI flash can not be allocated at SPIFI flash.
- Set the SPIFI to **Command Mode**.

Users can call functions loaded at RAM or internal flash to set the SPIFI flash to **Command Mode**, then erase flash and write data to SPIFI flash.

The demo project, `SDK_2.8.2_LPCXpresso54628_SPIFI\boards\lpcxpresso54628\demo_apps\spifi_usage\MDK\spifi_usage.uvmpw`, shows how to write data to SPIFI flash and read data from it.



```
int main(void)
{
    char ch;

    /* Init board hardware. */
    /* attach 12 MHz clock to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    spifi_flash_init();             /* Initialization for SPIFI flash */
    spifi_set_memory_mode();        /* Set SPIFI to memory mode */
    spifi_flash_func();             /* Call function allocated at SPIFI flash */

    spifi_set_command_mode();       /* set spifi to command mode */
    spifi_sector_program_test();    /* Call function allocated at RAM to write data to SPIFI flash */

    spifi_set_memory_mode();        /* Set SPIFI to memory mode */
    spifi_sector_read_test();       /* Read data from SPIFI flash */
```

Figure 10. Write data to SPIFI flash

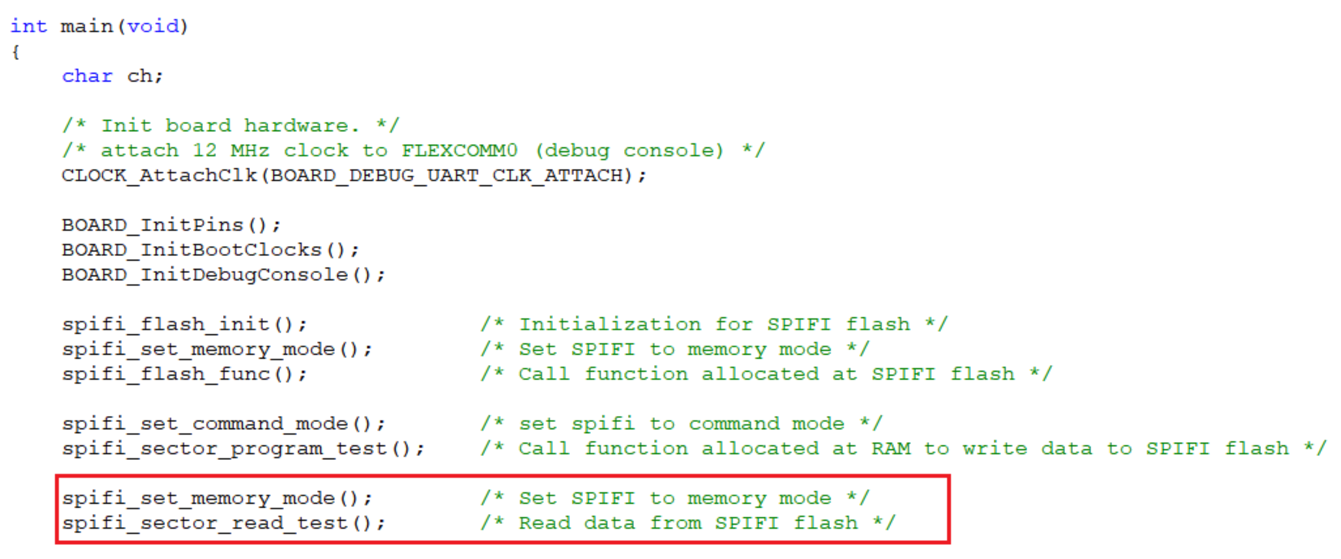
From the definition of `spifi_sector_program_test()` and the linker file, user can find that this function is loaded to RAM space.

The system will set the SPIFI work with command mode, erase one sector, and write data into SPIFI flash page by page.

4.2 Read data from SPIFI Flash

When SPIFI is in the **Memory Mode**, users can read SPIFI content directly with a pointer. Set the pointer points to the target address and get its value.

There is no restriction for reading data code location address, all of RAM, internal flash and SPIFI flash can work.



```
int main(void)
{
    char ch;

    /* Init board hardware. */
    /* attach 12 MHz clock to FLEXCOMM0 (debug console) */
    CLOCK_AttachClk(BOARD_DEBUG_UART_CLK_ATTACH);

    BOARD_InitPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    spifi_flash_init();             /* Initialization for SPIFI flash */
    spifi_set_memory_mode();        /* Set SPIFI to memory mode */
    spifi_flash_func();             /* Call function allocated at SPIFI flash */

    spifi_set_command_mode();       /* set spifi to command mode */
    spifi_sector_program_test();    /* Call function allocated at RAM to write data to SPIFI flash */

    spifi_set_memory_mode();        /* Set SPIFI to memory mode */
    spifi_sector_read_test();       /* Read data from SPIFI flash */
```

Figure 11. Read data test

5 SPIFI Flash performance

CoreMark is a simple and sophisticated benchmark, designed specifically to test the functionality of a processor core.

The project, located at *SDK_2.8.2_LPCXpresso54628_SPIFI_Features\boards\lpcxpresso54628\demo_apps\spifi_coremark\mdk*, is the SPIFI flash CoreMark test project. Users can run it with LPC54628 EVK to get the CoreMark score from the debug **uart0**. All of CoreMark information will be printed by **uart0**.

Click **Options for Target** → **Linker** → **Misc controls** and replace **--predefine="-D__coremark_spifi_flash__"** with **--predefine="-D__coremark_internal_flash__"** or **--predefine="-D__coremark_sram__"** to test the CoreMark score of internal flash or SRAM.

Table 2 describes the CoreMark score of the RAM, internal flash and SPIFI flash base on KEIL IDE.

Table 1. Test environment

LPC54628	Main Clock: 220 MHz, SPIFI Clock: 96 MHz
KEIL SETTING	V5.33, -Ofast
SDK	2.8.2

Table 2. CoreMark score of RAM, internal flash and SPIFI Flash base on KEIL IDE

Code location	Score (Iterations/Sec/MHz)
RAM	2.07
Internal Flash	1.65
SPIFI Flash	0.36

Date of release: February 25, 2021

Document identifier: AN13165