SPI-NAND driver for NXP FlexSPI

Rev. 1.2, 2023.9

Contents

Overview

At present, NXP MCUXpresso SDK only support FlexSPI NOR Flash. We add SPI NAND driver for FlexSPI controller using Macronix SPI NAND Flash MX35UF1GE4AC. Since MCUXpresso SDK has integrated LittleFS file system, we port the NFTL component from AliOS-Things to access NAND Flash better. As for the MXIC SPI NAND driver, We support most of the Flash command and feature such as software ECC, read recovery, continuous read and so on.

Development Platform

- Hardware Platform:
  - MIMXRT1170-EVK (B.v) + MX35UF1GE4AC-Z4I
- Software Platform
  - NXP MCUXpresso SDK (v. 2.13.1)

Prerequisite

- Flash: SPI NAND Flash
  - MX35 series
  - MX31 series (Lybra Flash)
- IDE: MCUXpresso v11.7.1
- Windows 10
- New added or modified files:

index	modified file
1	nxp-i.mxrt1170-spi-nand/board/peripherals.c
2	nxp-i.mxrt1170-spi-nand/board/peripherals.h
3	nxp-i.mxrt1170-spi-nand/board/pin_mux.c
4	nxp-i.mxrt1170-spi-nand/source/littlefs_shell.c

Table 1 modified file

index	new added/renamed file
1	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/flexspi_nand_flash.c
2	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/flexspi_nand_flash_ops.h
3	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/spi_nand_flash.c
4	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/spi_nand_flash.h
5	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/bch.c
6	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/bch.h
7	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/test/app_test.c
8	nxp-i.mxrt1170-spi-nand/flash/spi_nand_flash/test/ops_test.c
9	nxp-i.mxrt1170-spi-nand/nftl/nftl_conf.h
10	nxp-i.mxrt1170-spi-nand/nftl/nftl_partition.c
11	nxp-i.mxrt1170-spi-nand/nftl/nftl_partition.h
12	nxp-i.mxrt1170-spi-nand/nftl/nftl_util.c
13	nxp-i.mxrt1170-spi-nand/nftl/nftl_util.h
14	nxp-i.mxrt1170-spi-nand/nftl/nftl.c
15	nxp-i.mxrt1170-spi-nand/nftl/nftl.h
16	nxp-i.mxrt1170-spi-nand/source/littlefs_shell.h
17	nxp-i.mxrt1170-spi-nand/source/main.c

Table 2 new added/renamed file

Architecture

The project architecture diagram is shown as Figure 1. We provide NFTL component between SPI NAND driver and LittleFS, but users still can choose whether add the middleware or not. In the driver, the macro definition is located on */board/peripherals.h* (NFTL = 1).

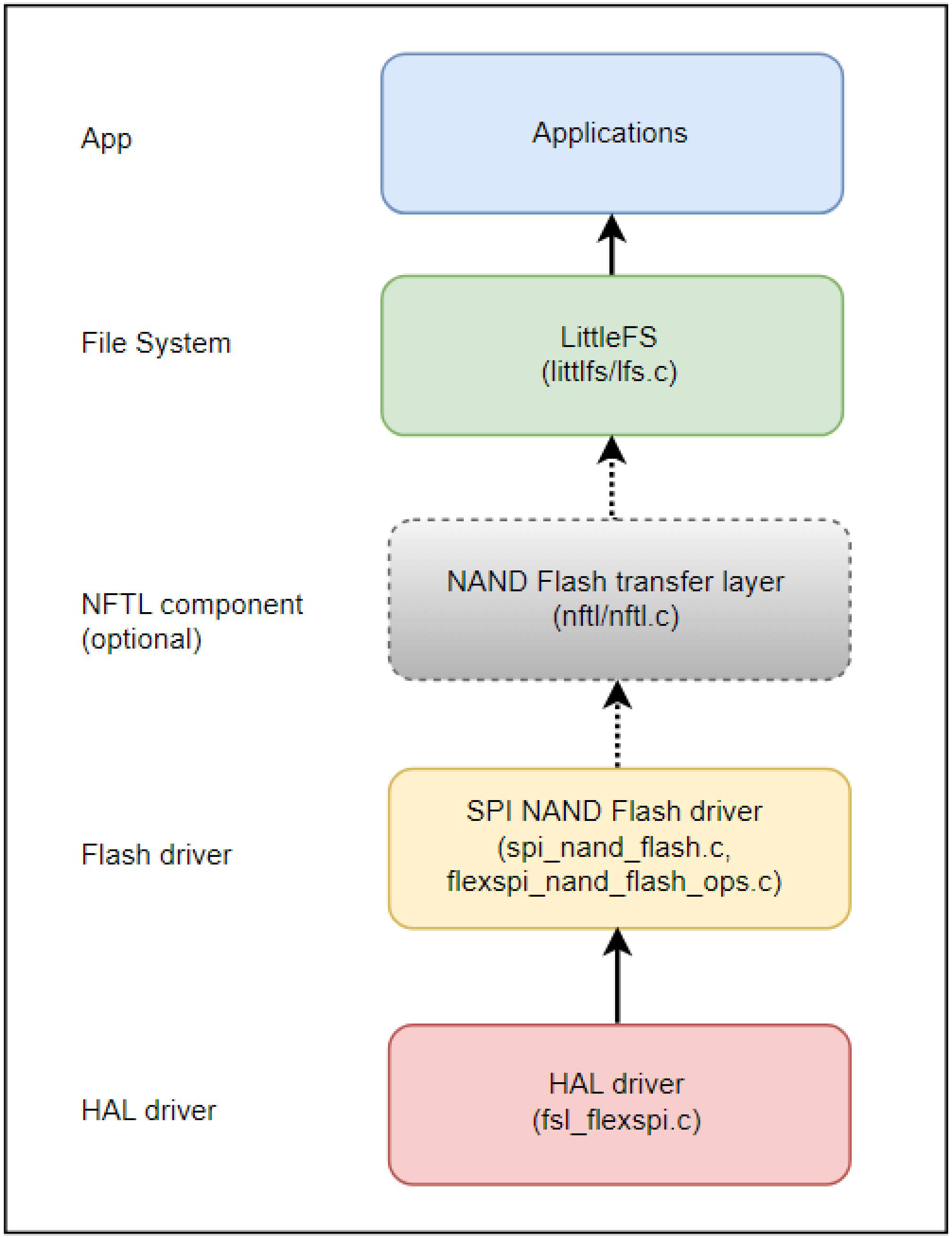


Figure 1 architecture diagram

APIs' Description

➢ SPI NAND Flash driver APIs

index	APIs	description
1	spi_nand_flash_init()	Initialization
2	spi_nand_read()	Read data from the array
3	spi_nand_read_oob()	Read data from oob area
4	spi_nand_write()	Write data to the array
5	spi_nand_write_oob()	Write data to the oob area
6	spi_nand_erase()	Erase blocks

Table 3 SPI NAND Flash driver APIs

➢ NFTL component APIs

index	APIs	description
1	nftl_init()	NFTL initialization
2	nftl_deinit()	NFTL deinitialization
3	nftl_flash_read()	Read Flash from logical block
4	nftl_flash_write()	Write Flash from logical block
5	nftl_flash_erase()	Erase a logical block
6	nftl_flash_syc()	Sync all buffered data onto Flash
7	nftl_flash_occupation()	Get the occupation information of a partation
8	nftl_flash_cleanup()	Cleanup a NFTL partition

Table 4 NFTL APIs

Driver Testcase

In order to test the driver and demo the APIs usage, we add some testcases for the SPI NAND driver *(spi_nand_flash.c)* and underlying driver *(flexspi_nand_flash.c)*.

➢ SPI NAND driver testcase

index	**testcase**	**description**
1	nand_unaligned_erase_blocks()	Test unaligned erase address
2	nand_program_read_small_data_sizes_test()	Test less than one page size read/write
4	nand_contiguous_erase_write_read_test()	Test contiguous 2 blocks read/write/erase
5	nand_unaligned_read_test()	Test read/write page with byte offset
6	nand_read_write_erase_test()	Basic read/write/erase testcase
7	nand_oob_read_write_test()	Test the OOB area read/write function

Table 5 SPI NAND driver testcase

➢ Underlying driver testcase

index	**testcase**	**description**
1	nand_read_id_test	Test read vendor ID function
2	nand_read_status_test()	Test read status register by send command code
4	nand_reset_test()	Test reset the device function
5	nand_dp_test()	Test enter deep power down mode
6	nand_cache_read_sequential_random_test()	Test cache read sequential/random mode

Table 6 underlying driver testcase

Hardware setup

- On RT1170, a QSPI Flash (IS25WP128) connect to FlexSPI1 controller as code Flash for boot. In this case, we connect the SPI NAND Flash to FlexSPI2 controller instead.
- **schematic modification**: connect SPI NAND Flash to FlexSPI2 2nd pin group GPIO_SD_B1[05:00]
    - GPIO_SD_B1_00 --> R1890 --> CS
    - GPIO_SD_B1_01 --> R370 --> SCLK
    - GPIO_SD_B1_02 --> R1892 --> D0
    - GPIO_SD_B1_03 --> R1893 --> D1
    - GPIO_SD_B1_00 --> R1894 --> D2
    - GPIO_SD_B1_00 --> R1895 --> D3
- If the SPI NAND Flash support 1.8V only, please connect the J55 jumper on RT1170 board. This operation will switch the output voltage to 1.8V from 3.3V.

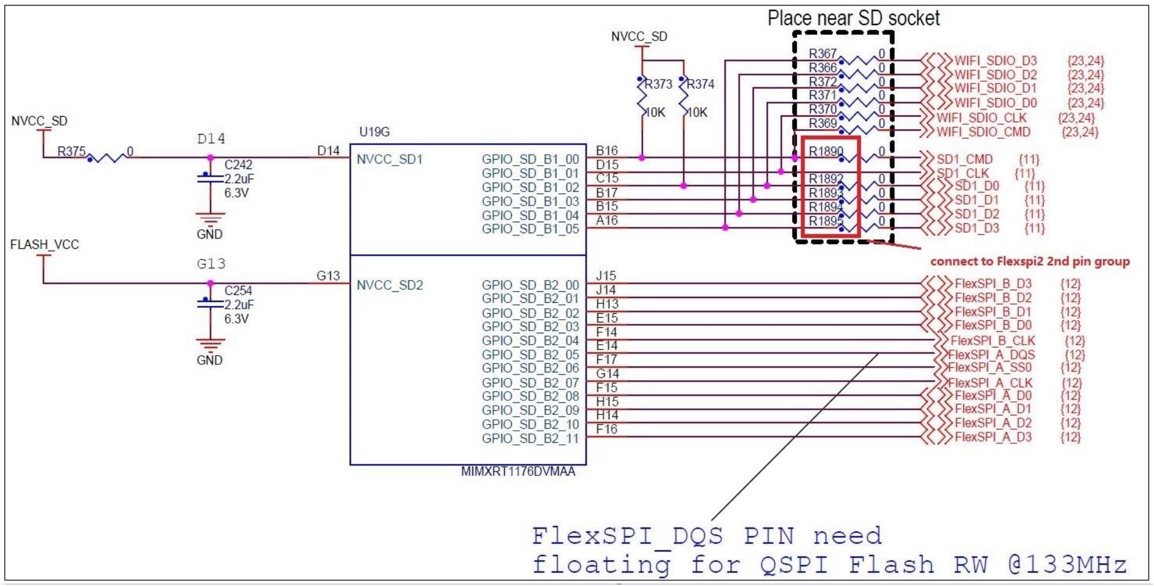

Place near SD socket
NVCC_SD
R373 10K
R374 10K
R367 R366 R372 R371 R370 R369
R1890 R1892 R1893 R1894 R1895
WIFI_SDIO_D3 WIFI_SDIO_D2 WIFI_SDIO_D1 WIFI_SDIO_D0 WIFI_SDIO_CLK WIFI_SDIO_CMD
SD1_CMD SD1_CLK SD1_D0 SD1_D1 SD1_D2 SD1_D3
connect to Flexspi2 2nd pin group
NVCC_SD R375 0
D14 C242 2.2uF 6.3V GND
FLASH_VCC G13 C254 2.2uF 6.3V GND
U19G NVCC_SD1 NVCC_SD2
GPIO_SD_B1_00 … GPIO_SD_B1_05
GPIO_SD_B2_00 … GPIO_SD_B2_11
FlexSPI_B_D3 FlexSPI_B_D2 FlexSPI_B_D1 FlexSPI_B_D0 FlexSPI_B_CLK FlexSPI_A_DQS FlexSPI_A_SS0 FlexSPI_A_CLK FlexSPI_A_D0 FlexSPI_A_D1 FlexSPI_A_D2 FlexSPI_A_D3
MIMXRT1176DVMAA
FlexSPI_DQS PIN need floating for QSPI Flash RW @133MHz

Figure 2 schematic modification

- **PCB layout location for Resistors**

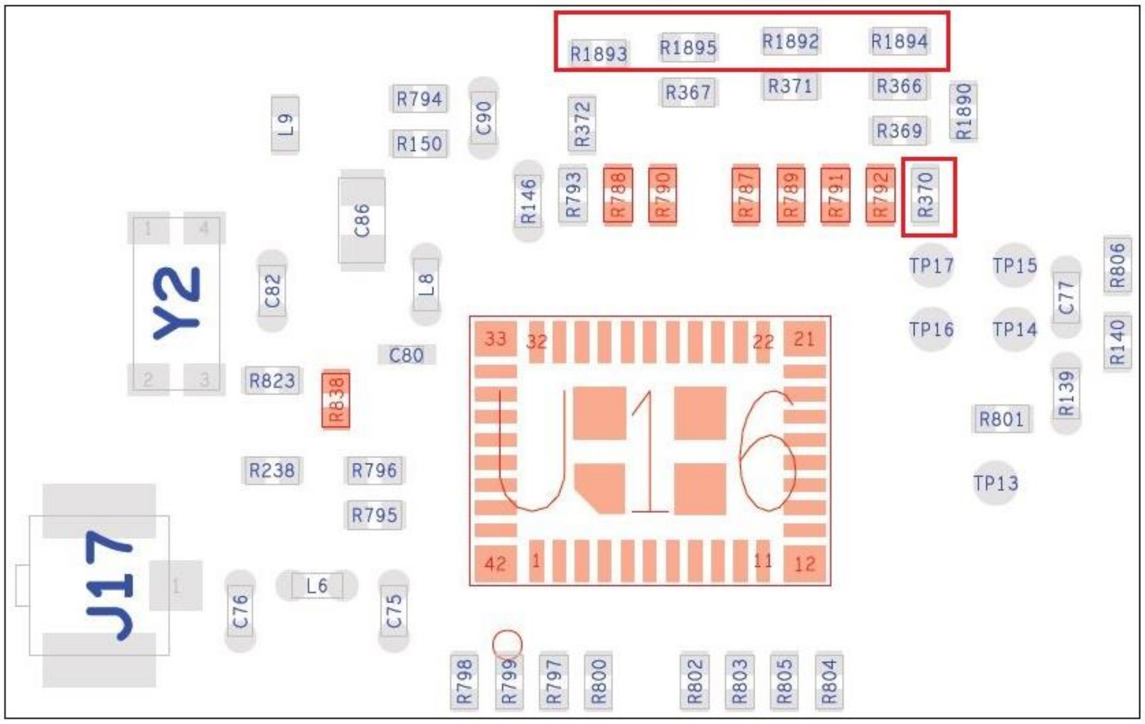


Figure 3 PCB layout location

Software modification highlight

We developed the SPI NAND Flash driver mainly inspired from MCUXpresso SDK (v 2.13.1) example project evkmimxrt1170_flexspi_nor_polling_transfer_cm7. And finally the SPI NAND Flash driver with NFTL middleware is ported into another example project evkmimxrt1170_littlefs_shell_cm7. The new added and modified file is shown as Figure 4.

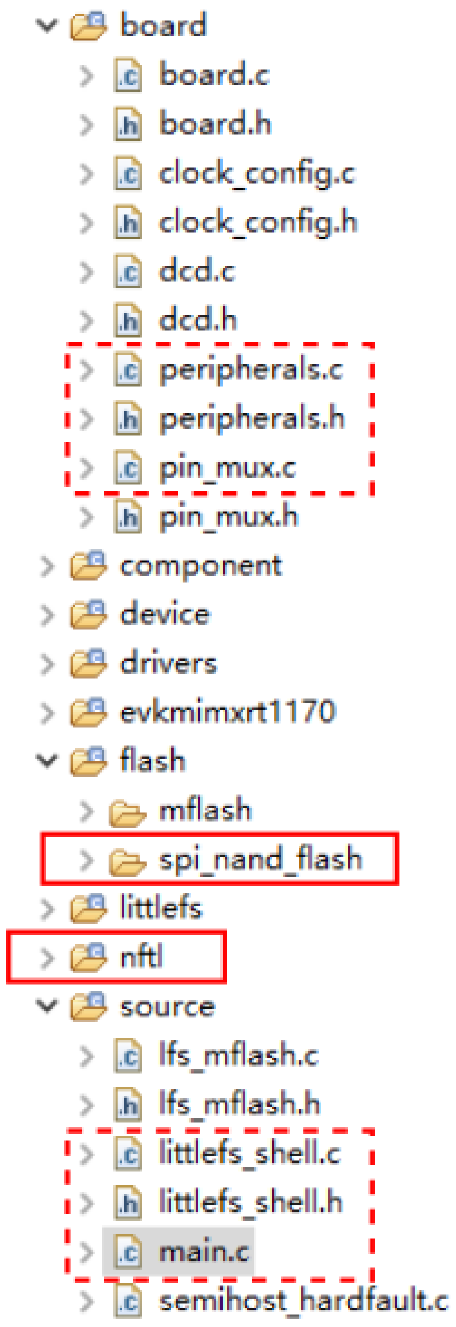
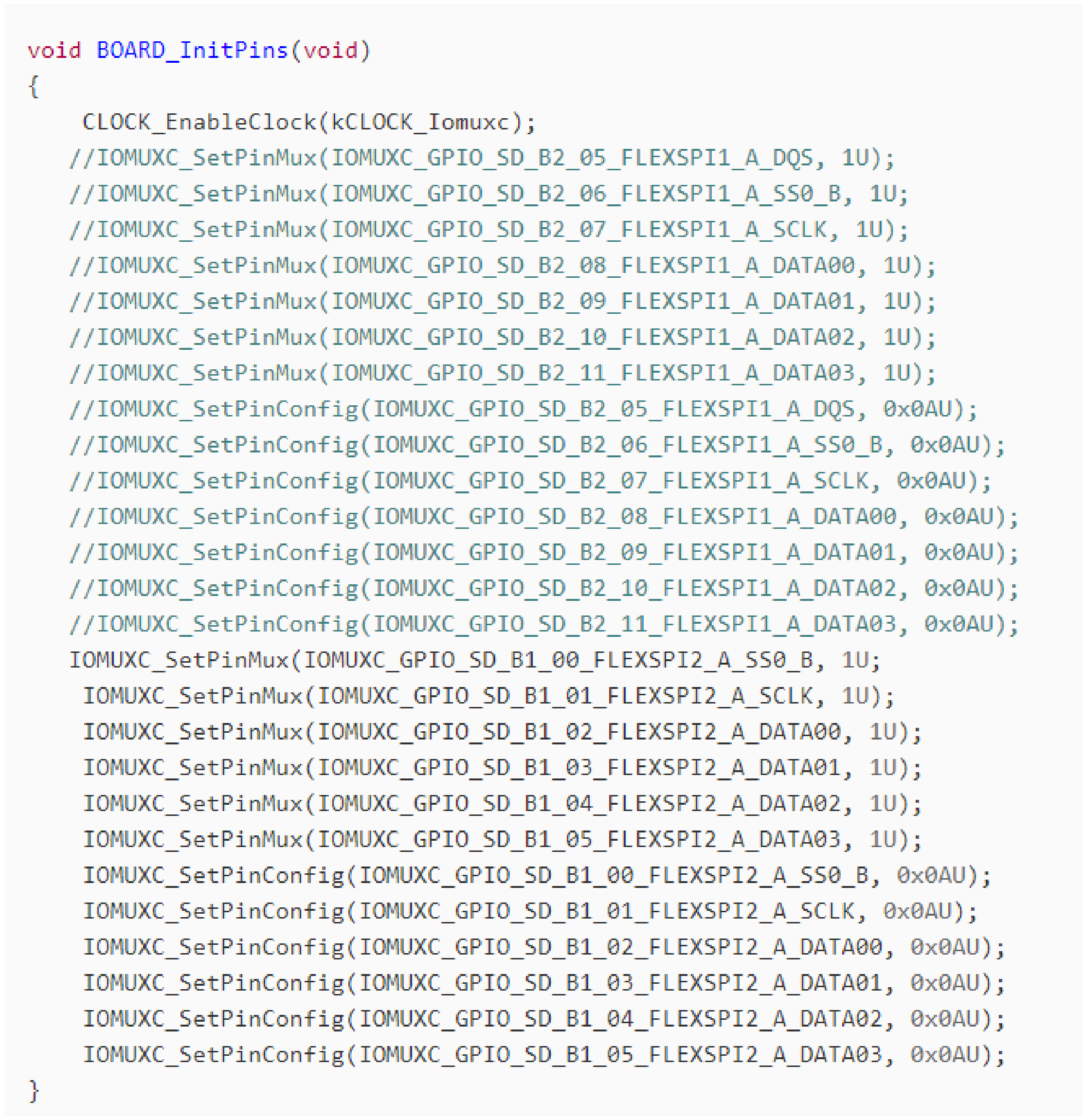


Figure 4 project workspace

1. initial the FlexSPI2 pin

    - board --> pin_mux.c:



```
void BOARD_InitPins(void)
{
    CLOCK_EnableClock(kCLOCK_Iomuxc);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_05_FLEXSPI1_A_DQS, 1U);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_06_FLEXSPI1_A_SS0_B, 1U;
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_07_FLEXSPI1_A_SCLK, 1U);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_08_FLEXSPI1_A_DATA00, 1U);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_09_FLEXSPI1_A_DATA01, 1U);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_10_FLEXSPI1_A_DATA02, 1U);
   //IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B2_11_FLEXSPI1_A_DATA03, 1U);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_05_FLEXSPI1_A_DQS, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_06_FLEXSPI1_A_SS0_B, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_07_FLEXSPI1_A_SCLK, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_08_FLEXSPI1_A_DATA00, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_09_FLEXSPI1_A_DATA01, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_10_FLEXSPI1_A_DATA02, 0x0AU);
   //IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B2_11_FLEXSPI1_A_DATA03, 0x0AU);
   IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_00_FLEXSPI2_A_SS0_B, 1U;
    IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_01_FLEXSPI2_A_SCLK, 1U);
    IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_02_FLEXSPI2_A_DATA00, 1U);
    IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_03_FLEXSPI2_A_DATA01, 1U);
    IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_04_FLEXSPI2_A_DATA02, 1U);
    IOMUXC_SetPinMux(IOMUXC_GPIO_SD_B1_05_FLEXSPI2_A_DATA03, 1U);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_00_FLEXSPI2_A_SS0_B, 0x0AU);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_01_FLEXSPI2_A_SCLK, 0x0AU);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_02_FLEXSPI2_A_DATA00, 0x0AU);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_03_FLEXSPI2_A_DATA01, 0x0AU);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_04_FLEXSPI2_A_DATA02, 0x0AU);
    IOMUXC_SetPinConfig(IOMUXC_GPIO_SD_B1_05_FLEXSPI2_A_DATA03, 0x0AU);
}
```

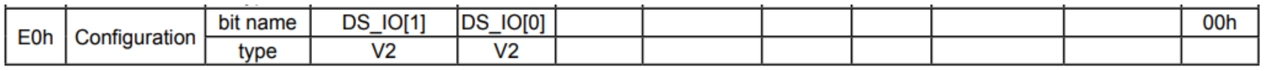
Figure 5 pin group initialization

2. initial the FlexSPI clock
   - flexspi_nand_flash_ops.h

```
//#define EXAMPLE_FLEXSPI FLEXSPI1
//#define EXAMPLE_FLEXSPI_AMBA_BASE FlexSPI1_AMBA_BASE
//#define EXAMPLE_FLEXSPI_CLOCK kCLOCK_Flexspi1
//#define EXAMPLE_FLEXSPI_RX_SAMPLE_CLOCK kFLEXSPI_ReadSampleClkLoopbackFromDqsPad

#define EXAMPLE_FLEXSPI FLEXSPI2
#define EXAMPLE_FLEXSPI_AMBA_BASE FlexSPI2_AMBA_BASE
#define EXAMPLE_FLEXSPI_CLOCK kCLOCK_Flexspi2
#define EXAMPLE_FLEXSPI_RX_SAMPLE_CLOCK kFLEXSPI_ReadSampleClkLoopbackInternally

static inline void flexspi_clock_init(void)
{
   //CLOCK_SetRootClockDiv(kCLOCK_Root_Flexspi1, 2);
   //CLOCK_SetRootClockMux(kCLOCK_Root_Flexspi1, 0);
   CLOCK_SetRootClockDiv(kCLOCK_Root_Flexspi2, 2);
    CLOCK_SetRootClockMux(kCLOCK_Root_Flexspi2, 0);
}
```

Figure 6 FlexSPI clock initialization

3. For the SPI NAND Flash driver, we have tested it and modified the PCB circuit on RT1170 EVB and there might be some hardware connection problem on it. If the 4IO read function fail, please refer to the Flash datasheet and configure the register as below.



E0h	Configuration	bit name	DS_IO[1]	DS_IO[0]							00h
		type	V2	V2							

Figure 7 configuration register

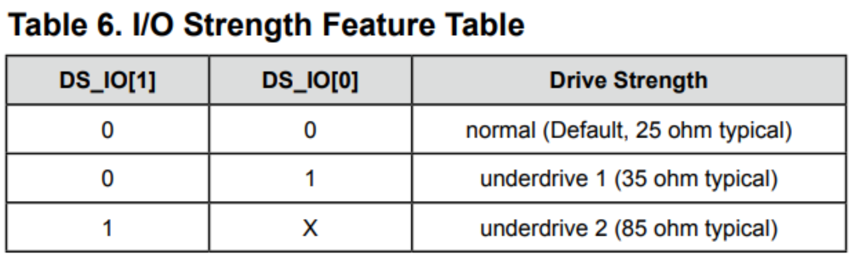


Table 6. I/O Strength Feature Table

DS_IO[1]	DS_IO[0]	Drive Strength
0	0	normal (Default, 25 ohm typical)
0	1	underdrive 1 (35 ohm typical)
1	X	underdrive 2 (85 ohm typical)

Table 7 I/O strength feature

4. Continuous read mode
   - The SPI NAND Flash supports continuous read mode, the driver will enter the continuous read mode in default when the read function is called.

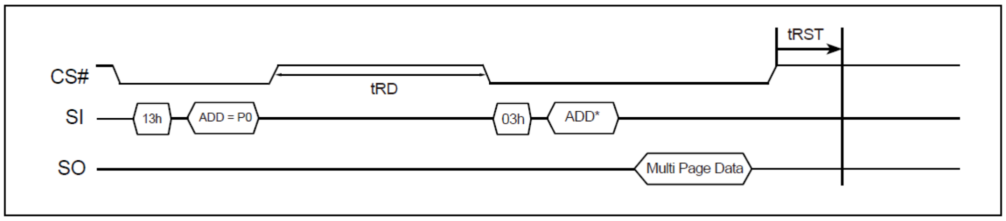
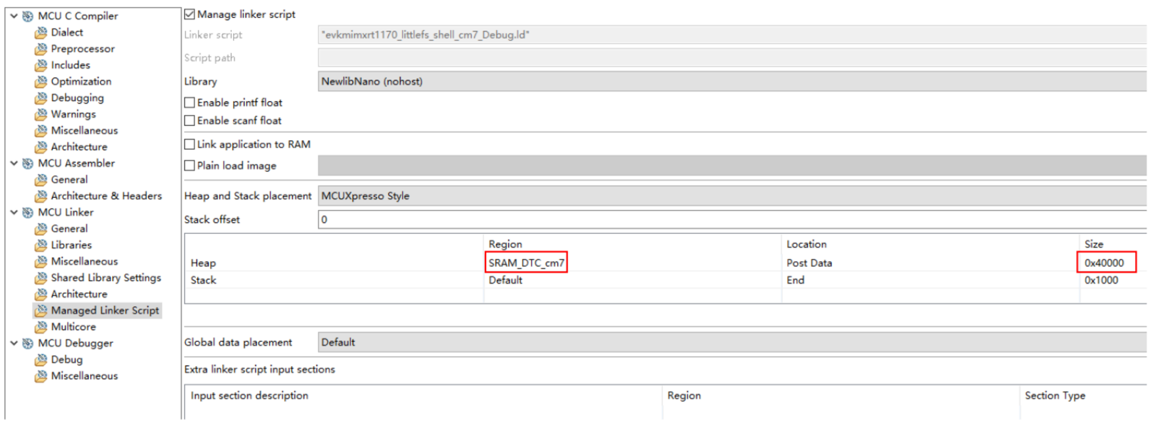


Figure 8 continuous read waveform

5. SPI NOR Compatible Command
   - The SPI NAND Flash provides SPI NOR like read command, specific information please refer to the Flash datasheet. And this feature is OTP (One Time Programmable), the underlying driver provide the API (*flexspi_nand_spi_nor_enable()*) to enable it, users should call the API in initialization (*spi_nand_flash_init()*), then set the macro definition (SPI_NOR_EN = 1) to enable it （*flexspi_nand_flash_ops.h*）.

6. In the NFTL partition initialization, it needs a 128KB heap to store the logic block information. The default heap size for this example project is 1KB, we can do some settings on MCUXpresso IDE to fix it. Firstly, modify the heap size to 0x4000, then change the heap region to SRAM_DTC_cm7 (Figure 9). And the total built binary file exceeds the SRAM_DTC (256KB) size, so it needs to select the SRAM_OC1 as default RAM (Figure 10).



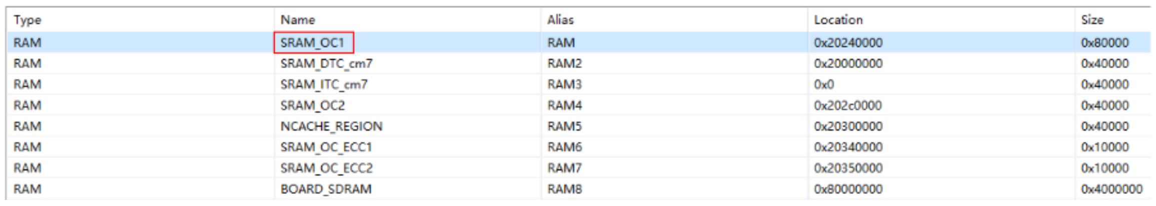
[✓] Manage linker script
Linker script: "evkmimxrt1170_littlefs_shell_cm7_Debug.ld"
Script path:
Library: NewlibNano (nohost)
[ ] Enable printf float
[ ] Enable scanf float
[ ] Link application to RAM
[ ] Plain load image
Heap and Stack placement: MCUXpresso Style
Stack offset: 0

	Region	Location	Size
Heap	SRAM_DTC_cm7	Post Data	0x40000
Stack	Default	End	0x1000

Global data placement: Default
Extra linker script input sections

Input section description	Region	Section Type

Figure 9 heap and stack placement



Type	Name	Alias	Location	Size
RAM	SRAM_OC1	RAM	0x20240000	0x80000
RAM	SRAM_DTC_cm7	RAM2	0x20000000	0x40000
RAM	SRAM_ITC_cm7	RAM3	0x0	0x40000
RAM	SRAM_OC2	RAM4	0x202c0000	0x40000
RAM	NCACHE_REGION	RAM5	0x20300000	0x40000
RAM	SRAM_OC_ECC1	RAM6	0x20340000	0x10000
RAM	SRAM_OC_ECC2	RAM7	0x20350000	0x10000
RAM	BOARD_SDRAM	RAM8	0x80000000	0x4000000

Figure 10 MCU settings

Testbench

In order to verify the correctness of driver and hardware controller, we provide complete testbench. The framework of the testbench is shown in Figure 11.

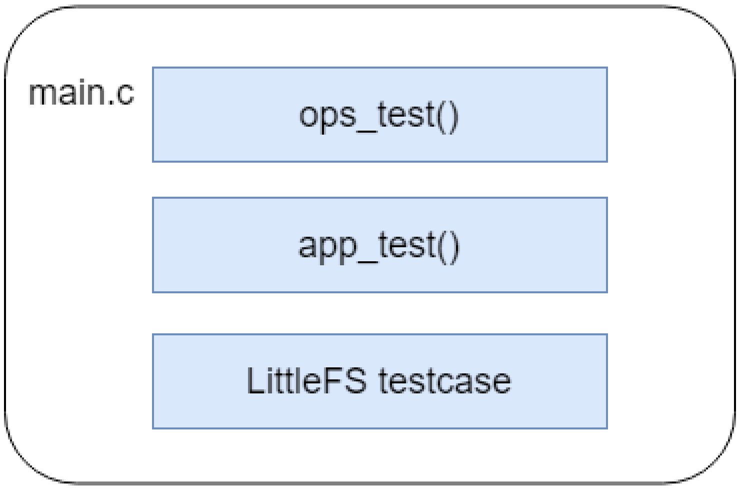
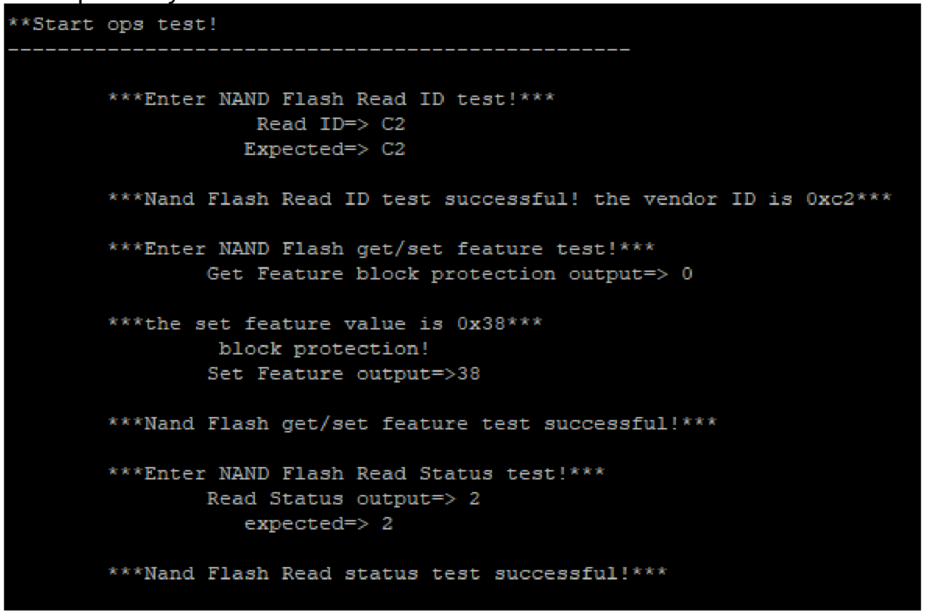


Figure 11 framework of testbench

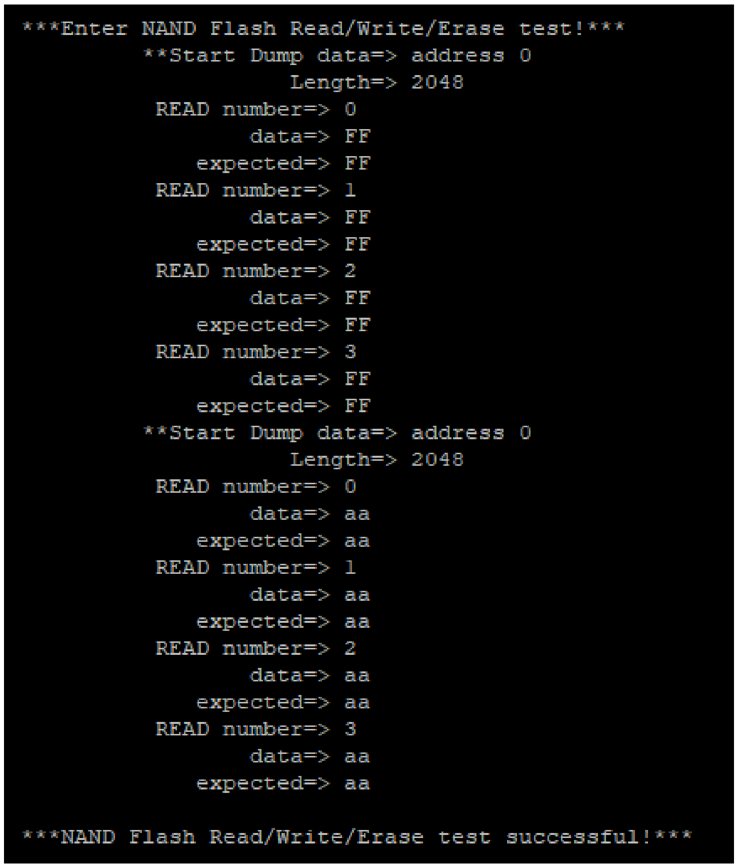
- Ops_test: this part tests the underlying driver functions in ops layer, like *flexspi_nand_read_id()*, *flexspi_nand_get_feature()* and etc.
- app_test: this part tests the SPI-NAND driver funcitons, like read, write and erase operations with 1, 2, 4 IO read mode.
- LittleFS testcase: this part tests the LittleFS operations with NFTL and SPI-NAND driver, like format, mount, write and etc.

We split the testcases into LittleFS testcase, SPI NAND driver testcase and underlying driver testcase. When set the SPI_NAND_TEST_MODE 1 in *main.c*, the program will enter driver testing mode. Then the SPI NAND and underlying driver testcases execute sequentially.



```
**Start ops test!
--------------------------------------------------

        ***Enter NAND Flash Read ID test!***
                    Read ID=> C2
                   Expected=> C2

        ***Nand Flash Read ID test successful! the vendor ID is 0xc2***

        ***Enter NAND Flash get/set feature test!***
                Get Feature block protection output=> 0

        ***the set feature value is 0x38***
                 block protection!
                Set Feature output=>38

        ***Nand Flash get/set feature test successful!***

        ***Enter NAND Flash Read Status test!***
                Read Status output=> 2
                   expected=> 2

        ***Nand Flash Read status test successful!***
```

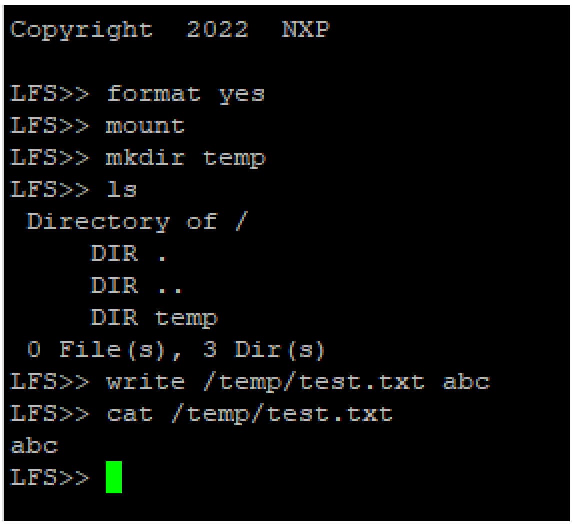
Figure 12 underlying driver testcase



```
***Enter NAND Flash Read/Write/Erase test!***
         **Start Dump data=> address 0
                    Length=> 2048
          READ number=> 0
                 data=> FF
             expected=> FF
          READ number=> 1
                 data=> FF
             expected=> FF
          READ number=> 2
                 data=> FF
             expected=> FF
          READ number=> 3
                 data=> FF
             expected=> FF
         **Start Dump data=> address 0
                    Length=> 2048
          READ number=> 0
                 data=> aa
             expected=> aa
          READ number=> 1
                 data=> aa
             expected=> aa
          READ number=> 2
                 data=> aa
             expected=> aa
          READ number=> 3
                 data=> aa
             expected=> aa

***NAND Flash Read/Write/Erase test successful!***
```

Figure 13 SPI-NAND driver testcase

Before running the LittleFS testcase, user should erase the flash first, calling *spi_nand_erase()* in *spi_nand_flash.c* . This SDK example project only support IDE online test, because the firmware is updated into OCM instead of boot NOR Flash. After the shell shows via UART, user can refer to Figure 10 to have a simple test for LittleFS operation on SPI NAND Flash.



```
Copyright  2022  NXP

LFS>> format yes
LFS>> mount
LFS>> mkdir temp
LFS>> ls
 Directory of /
     DIR .
     DIR ..
     DIR temp
 0 File(s), 3 Dir(s)
LFS>> write /temp/test.txt abc
LFS>> cat /temp/test.txt
abc
LFS>>
```

Figure 14 shell test reference

Revision History

Date	version	Description
7/2023	1.0	Basic read/write/erase function
7/2023	1.1	Add NFTL and LittleFS middleware
9/2023	1.2	1. Add more command, like sequential/random read, deep down, read/write BBM etc. 2. Add continuous read, read recovery, software ECC, hardware randomizer feature 3. Add testbench for SPI-NAND and underlying driver

MACRONIX INTERNATIONAL CO., LTD.