LittleFS MXIC Middleware

Hardware and Software Requirement:

1. Renesas FSP V4.0.0 or later
2. Renesas RA Series Board
3. MXIC Nor and SPI Nand Flash

Middleware Introduction

1. In Renesas FSP, Users can add LittleFS to their project, and the filesystem memory is internal flash. The middleware can connect LittleFS with external MXIC nor flash.
2. This middleware can be built upon both QSPI and OSPI driver. And if you choose QSPI, you can use SPI Nand Flash as the filesystem memory.
3. When user select SPI Nand Flash in LittleFS, user can also enable NFTL, For more information about NFTL, please refer NFTL_Introduction.pdf.
4. When add NFTL to the project, user should set as large a heap and stack as possible. For example, here we set Stack = 0x8000, Heap = 0x25000. (The Stack and Heap size also depends on your application, especially Heap.)

Sample code

```
#include "hal_data.h"

#if NFTL_ENABLE
#include "../ra/fsp/src/rm_littlefs_mxic_qspi_flash/spi_nand/spi_nand_app.h"
extern MxChip Mxic;
#endif

void hal_entry (void)
{
    module_init(); // Init QSPI or OSPI controller

#if NFTL_ENABLE
    int ret = 0;
    ret = BSP_SPI_Nand_Init(&Mxic, &g_qspi0);
    SEGGER_RTT_printf(0, "ID =%x%x\r\n", Mxic.Id[0],Mxic.Id[1]);
    ret = BSP_SPI_Erase_Block(&Mxic, 0, 272);
    SEGGER_RTT_printf(0, "Erase Ret = %d\r\n", ret);
    nftl_init();
    SEGGER_RTT_printf(0, "Init Successful\r\n");
#endif
    uint8_t rbuffer[100];
    uint8_t wbuffer[100];
    g_rm_littlefs0.p_api->open(g_rm_littlefs0.p_ctrl, g_rm_littlefs0.p_cfg);

    int err = lfs_format(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg);
    if(FSP_SUCCESS != err)
    {
       printf("** littleFs format failed **\r\n");
    }
    err = lfs_mount(&g_rm_littlefs0_lfs, &g_rm_littlefs0_lfs_cfg);
    if(FSP_SUCCESS != err)
    {
       printf("** littleFs Mount failed **\r\n");
    }

    err = lfs_mkdir(&g_rm_littlefs0_lfs, "potato");
    if(FSP_SUCCESS != err)
    {
       printf("** littleFs mkdir failed **\r\n");
    }

    uint32_t boot_count = 0;
    uint32_t r          = 0;
    lfs_file_t file;
    lfs_file_open(&g_rm_littlefs0_lfs, &file, "burito", LFS_O_RDWR |
LFS_O_CREAT);
    lfs_file_read(&g_rm_littlefs0_lfs, &file, &boot_count, sizeof(boot_count));

    // update boot count
    boot_count += 1;
    lfs_file_rewind(&g_rm_littlefs0_lfs, &file);
    lfs_file_write(&g_rm_littlefs0_lfs, &file, &boot_count, sizeof(boot_count));
    // remember the storage is not updated until the file is closed successfully
    lfs_file_close(&g_rm_littlefs0_lfs, &file);

    lfs_file_open(&g_rm_littlefs0_lfs, &file, "burito", LFS_O_RDWR |
LFS_O_CREAT);
    lfs_file_read(&g_rm_littlefs0_lfs, &file, &r, sizeof(r));
    lfs_file_close(&g_rm_littlefs0_lfs, &file);
    if(r != 1)
    {
        printf("read data error\r\n");
    }

    size_t size = 0;
    lfs_file_t file1;
    lfs_file_open(&g_rm_littlefs0_lfs, &file1, "tomato", LFS_O_CREAT |
LFS_O_WRONLY);
    size = strlen("Hello World!\n");
    memcpy(wbuffer, "Hello World!\n", size);
    lfs_file_write(&g_rm_littlefs0_lfs, &file1, wbuffer, size);
    lfs_file_close(&g_rm_littlefs0_lfs, &file1);

    lfs_file_open(&g_rm_littlefs0_lfs, &file1, "tomato", LFS_O_CREAT |
LFS_O_RDONLY);
    lfs_file_read(&g_rm_littlefs0_lfs, &file1, rbuffer, size);
    err = memcmp(rbuffer, wbuffer, size);
    if(FSP_SUCCESS != err)
    {
        printf("** littleFs file test failed **\r\n");
    }
    lfs_file_close(&g_rm_littlefs0_lfs, &file1);

    // release any resources we were using
    lfs_unmount(&g_rm_littlefs0_lfs);

    g_rm_littlefs0.p_api->close(g_rm_littlefs0.p_ctrl);
    g_qspi0.p_api->close(g_qspi0.p_ctrl); //g_ospi0.p_api-
>close(g_ospi0.p_ctrl);

    while (1)
    {

    }
}
```