需求简介

RT1064相比RT1060最大的区别就在于其内部封装了一颗4MB QSPI芯片，但4MB的空间对有些客户来说不一定就足够，所以需要额外添加一个空间更大的外置QSPI来存放application代码，从而在RT1064成功启动后能够跳转过去执行。 简单来讲就是内置的4MB QSPI存放second bootloader代码，在RT1064启动后，跳转到空间更大的外置QSPI运行application代码（如下图所示）。

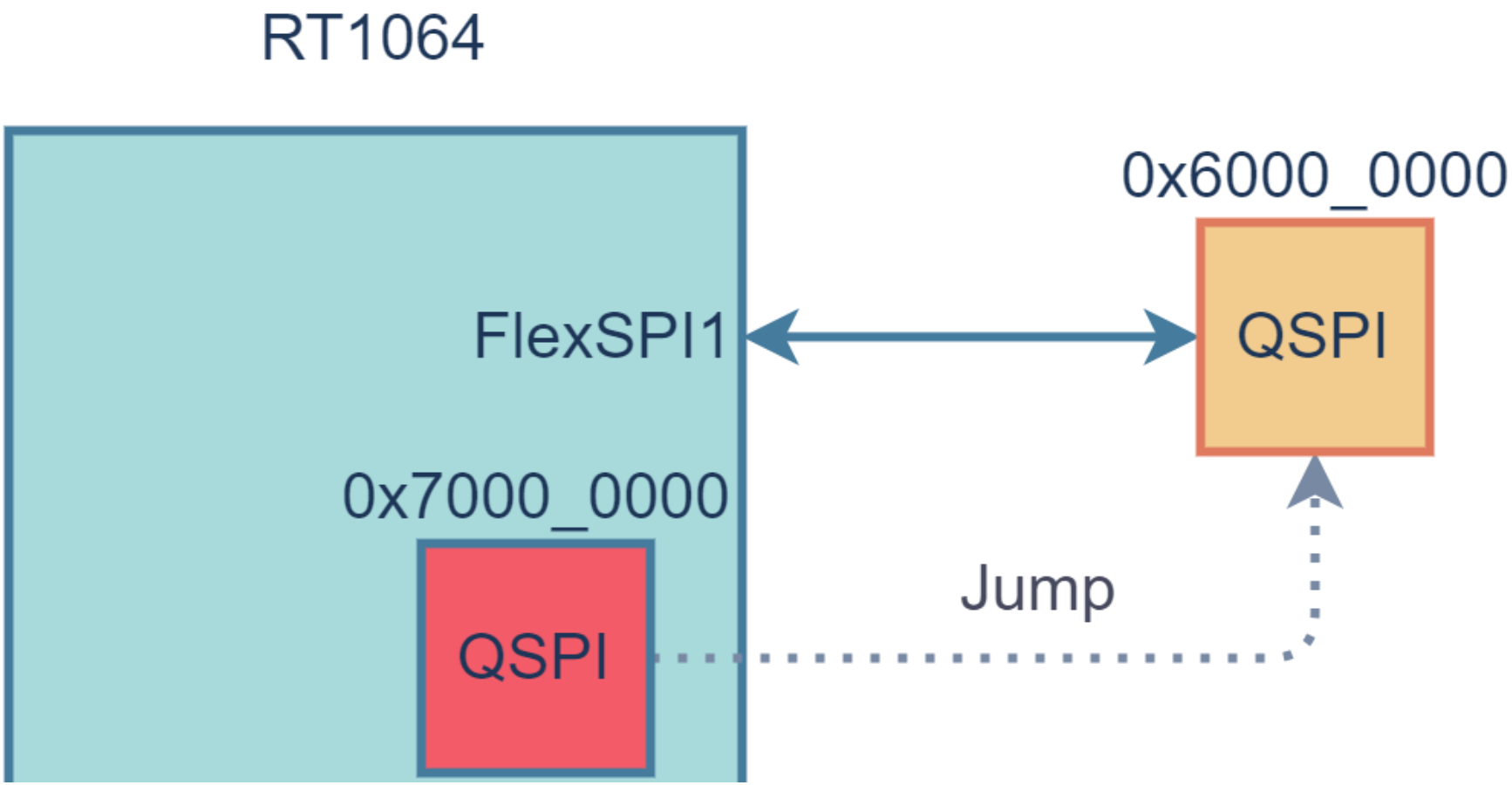


图1 程序跳转流程

实现流程

1.硬件开发板

- MIMXRT1064-EVK含有外置QSPI (IS25WP064AJBLE)，满足硬件测试要求；

2. 软件代码准备

2.1) 创建second bootloader代码

second bootloader代码以evkmimxrt1064_flexspi_nor_polling_transfer工程为基础，用于实现初始化连接外置QSPI的FleSPI1接口和代码跳转操作。

```
#include "fsl_flexspi.h"
#include "app.h"
#include "fsl_debug_console.h"
#include "fsl_cache.h"

#include "pin_mux.h"
#include "clock_config.h"
```

```
#include "board.h"
#include "fsl_common.h"
/*******************************************************************************
 * Definitions
 ******************************************************************************/
#define SET_THUMB_ADDRESS(x) (x | 0x1)

#define Application_offset 0x2000

// typedef for function used to do the jump to the application

/*******************************************************************************
 * Prototypes
 ******************************************************************************/
typedef void (*app_entry_t)(void);

/*******************************************************************************
 * Variables
 ******************************************************************************/
/* Program data buffer should be 4-bytes alignment, which can avoid busfault due to this memory region is configured as
   Device Memory by MPU. */
app_entry_t appEntry = 0;

SDK_ALIGN(static uint8_t s_nor_program_buffer[256], 4);
static uint8_t s_nor_read_buffer[256];

extern status_t flexspi_nor_flash_erase_sector(FLEXSPI_Type *base, uint32_t address);
extern status_t flexspi_nor_flash_page_program(FLEXSPI_Type *base, uint32_t dstAddr, const uint32_t *src);
extern status_t flexspi_nor_get_vendor_id(FLEXSPI_Type *base, uint8_t *vendorId);
extern status_t flexspi_nor_enable_quad_mode(FLEXSPI_Type *base);
extern status_t flexspi_nor_erase_chip(FLEXSPI_Type *base);
extern void flexspi_nor_flash_init(FLEXSPI_Type *base);
//Jump to application code
static void JumpToAddr(uint32_t appaddr);
```

```
static void JumpToAddr(uint32_t appaddr)
{
    appaddr += FlexSPI_AMBA_BASE;

    // Point entry point address to entry point call function
    appEntry = (app_entry_t)(SET_THUMB_ADDRESS((*(uint32_t *)(appaddr + 4))));

    PRINTF("\r\nApplication address is 0x%p\r\n",appaddr);

    // Set the VTOR to the application vector table address.
    SCB->VTOR = (uint32_t)appaddr;

    // Set stack pointers to the application stack pointer.
    __set_MSP((uint32_t)(*(uint32_t *)(appaddr)));
    __set_PSP((uint32_t)(*(uint32_t *)(appaddr)));

    // Jump to main app entry point
    appEntry();
}

/*******************************************************************************
 * Code
 ******************************************************************************/
flexspi_device_config_t deviceconfig = {
    .flexspiRootClk       = 120000000,
    .flashSize            = FLASH_SIZE,
    .CSIntervalUnit       = kFLEXSPI_CsIntervalUnit1SckCycle,
    .CSInterval           = 2,
    .CSHoldTime           = 3,
    .CSSetupTime          = 3,
    .dataValidTime        = 0,
    .columnspace          = 0,
    .enableWordAddress    = 0,
    .AWRSeqIndex          = 0,
    .AWRSeqNumber         = 0,
    .ARDSeqIndex          = NOR_CMD_LUT_SEQ_IDX_READ_FAST_QUAD,
```

```
    .ARDSeqNumber         = 1,
    .AHBWriteWaitUnit     = kFLEXSPI_AhbWriteWaitUnit2AhbCycle,
    .AHBWriteWaitInterval = 0,
};

const uint32_t customLUT[CUSTOM_LUT_LENGTH] = {
    /* Normal read mode -SDR */
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_NORMAL] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x03, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_NORMAL + 1] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_READ_SDR, kFLEXSPI_1PAD, 0x04, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Fast read mode - SDR */
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_FAST] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x0B, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_FAST + 1] = FLEXSPI_LUT_SEQ(
        kFLEXSPI_Command_DUMMY_SDR, kFLEXSPI_1PAD, 0x08, kFLEXSPI_Command_READ_SDR, kFLEXSPI_1PAD, 0x04),

    /* Fast read quad mode - SDR */
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_FAST_QUAD] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x6B, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
    [4 * NOR_CMD_LUT_SEQ_IDX_READ_FAST_QUAD + 1] = FLEXSPI_LUT_SEQ(
        kFLEXSPI_Command_DUMMY_SDR, kFLEXSPI_4PAD, 0x08, kFLEXSPI_Command_READ_SDR, kFLEXSPI_4PAD, 0x04),

    /* Read extend parameters */
    [4 * NOR_CMD_LUT_SEQ_IDX_READSTATUS] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x81, kFLEXSPI_Command_READ_SDR, kFLEXSPI_1PAD, 0x04),

    /* Write Enable */
    [4 * NOR_CMD_LUT_SEQ_IDX_WRITEENABLE] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x06, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Erase Sector  */
    [4 * NOR_CMD_LUT_SEQ_IDX_ERASESECTOR] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0xD7, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
```

```
    /* Page Program - single mode */
    [4 * NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM_SINGLE] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x02, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
    [4 * NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM_SINGLE + 1] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_WRITE_SDR, kFLEXSPI_1PAD, 0x04, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Page Program - quad mode */
    [4 * NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM_QUAD] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x32, kFLEXSPI_Command_RADDR_SDR, kFLEXSPI_1PAD, 0x18),
    [4 * NOR_CMD_LUT_SEQ_IDX_PAGEPROGRAM_QUAD + 1] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_WRITE_SDR, kFLEXSPI_4PAD, 0x04, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Read ID */
    [4 * NOR_CMD_LUT_SEQ_IDX_READID] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x9F, kFLEXSPI_Command_READ_SDR, kFLEXSPI_1PAD, 0x04),

    /* Enable Quad mode */
    [4 * NOR_CMD_LUT_SEQ_IDX_WRITESTATUSREG] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x01, kFLEXSPI_Command_WRITE_SDR, kFLEXSPI_1PAD, 0x04),

    /* Enter QPI mode */
    [4 * NOR_CMD_LUT_SEQ_IDX_ENTERQPI] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x35, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Exit QPI mode */
    [4 * NOR_CMD_LUT_SEQ_IDX_EXITQPI] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_4PAD, 0xF5, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),

    /* Read status register */
    [4 * NOR_CMD_LUT_SEQ_IDX_READSTATUSREG] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0x05, kFLEXSPI_Command_READ_SDR, kFLEXSPI_1PAD, 0x04),

    /* Erase whole chip */
    [4 * NOR_CMD_LUT_SEQ_IDX_ERASECHIP] =
        FLEXSPI_LUT_SEQ(kFLEXSPI_Command_SDR, kFLEXSPI_1PAD, 0xC7, kFLEXSPI_Command_STOP, kFLEXSPI_1PAD, 0),
};
```

```
int main(void)
{
    uint32_t i = 0;
    status_t status;
    uint8_t vendorID = 0;
    char ch;


    BOARD_ConfigMPU();
    BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitDebugConsole();

    flexspi_nor_flash_init(EXAMPLE_FLEXSPI);

    PRINTF("\r\nFLEXSPI example started!\r\n");
    PRINTF("\r\nReset status reghister is 0x%x \r\n",SRC->SRSR);
    /* Get vendor ID. */
    status = flexspi_nor_get_vendor_id(EXAMPLE_FLEXSPI, &vendorID);
    if (status != kStatus_Success)
    {
        return status;
    }
    PRINTF("Vendor ID: 0x%x\r\n", vendorID);

#if !(defined(XIP_EXTERNAL_FLASH))
    /* Erase whole chip . */
    PRINTF("Erasing whole chip over FlexSPI...\r\n");

    status = flexspi_nor_erase_chip(EXAMPLE_FLEXSPI);
    if (status != kStatus_Success)
    {
        return status;
```

```
    }
    PRINTF("Erase finished !\r\n");

#endif

    /* Enter quad mode. */
    status = flexspi_nor_enable_quad_mode(EXAMPLE_FLEXSPI);
    if (status != kStatus_Success)
    {
        return status;
    }

    //DCACHE_InvalidateByRange(EXAMPLE_FLEXSPI_AMBA_BASE, FLASH_PAGE_SIZE);

    while (1)
    {
        ch = GETCHAR();
        // when input 's' charter, it will jump to application code soon
        if(ch == 's')
         JumpToAddr(Application_offset);
        else
         PUTCHAR(ch);
    }

    while (1)
    {
    }
}
```

2.2) 创建application code

选用SDK软件库中的evkmimxrt1064_iled_blinky作为application code，代码修改步骤如下所示。

- 在BOARD_InitBootClocks（）函数中，注释掉影响连接外置QSPI的FleSPI1接口时钟运行的代码；

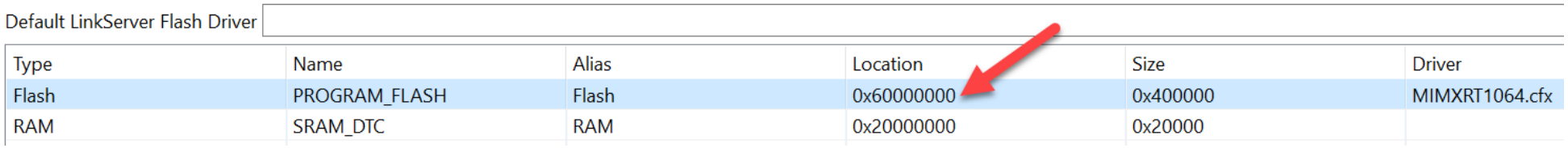
```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

   /* Disable Semc clock gate. */
    CLOCK_DisableClock(kCLOCK_Semc);
    /* Set SEMC_PODF. */
    CLOCK_SetDiv(kCLOCK_SemcDiv, 7);
    /* Set Semc alt clock source. */
    CLOCK_SetMux(kCLOCK_SemcAltMux, 0);
    /* Set Semc clock source. */
    CLOCK_SetMux(kCLOCK_SemcMux, 0);

//    /* Disable Flexspi clock gate. */
//    CLOCK_DisableClock(kCLOCK_FlexSpi);
//    /* Set FLEXSPI_PODF. */
//    CLOCK_SetDiv(kCLOCK_FlexspiDiv, 1);
//    /* Set Flexspi clock source. */
//    CLOCK_SetMux(kCLOCK_FlexspiMux, 3);
#endif

~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

- 外置QSPI对应的地址区域开始于0x6000_0000，所以需调整代码工程的存储空间布局（如下所示），保存后再次编译工程生成BIN文件；



Default LinkServer Flash Driver

Type	Name	Alias	Location	Size	Driver
Flash	PROGRAM_FLASH	Flash	0x60000000	0x400000	MIMXRT1064.cfx
RAM	SRAM_DTC	RAM	0x20000000	0x20000	

RAM	SRAM_ITC	RAM2	0x0	0x20000
RAM	SRAM_OC	RAM3	0x20200000	0xc0000
RAM	BOARD_SDRAM	RAM4	0x80000000	0x1e00000
RAM	NCACHE_REGION	RAM5	0x81e00000	0x200000

图2 存储空间布局

3.烧录appliction 代码到外置QSPI

打开J-Flash,创建新工程，选择MIMXRT1064（如下图所示），将上面生成的evkmimxrt1064_iled_blinky工程的BIN文件拖拽到操作界面，

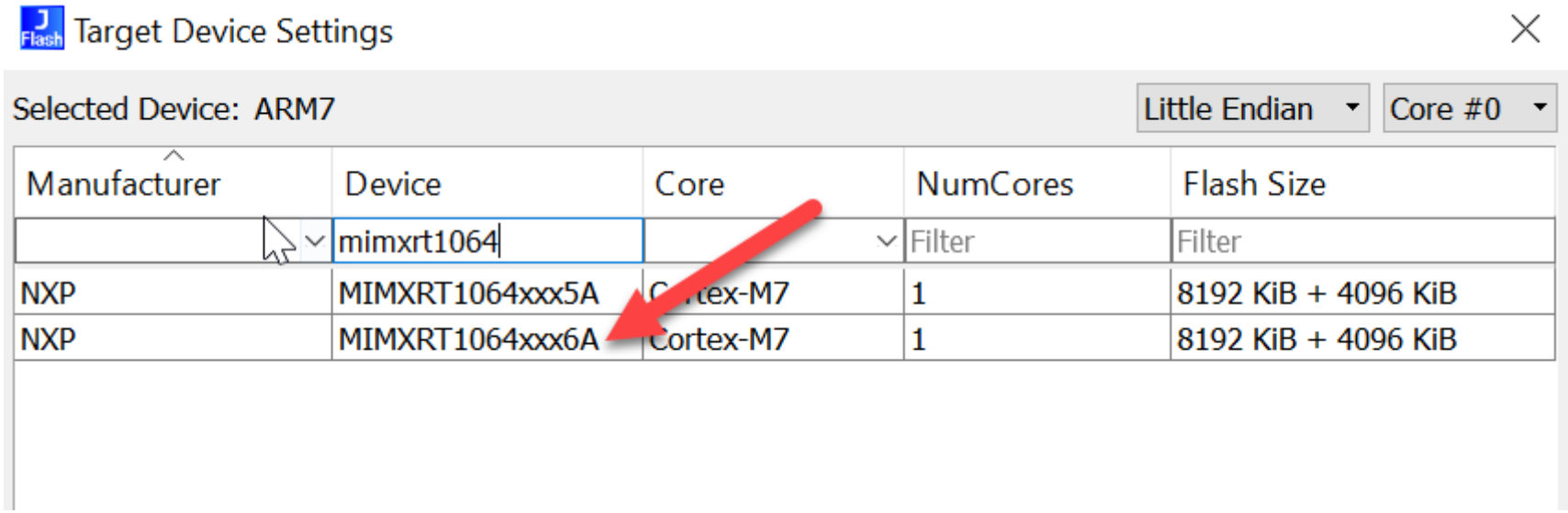
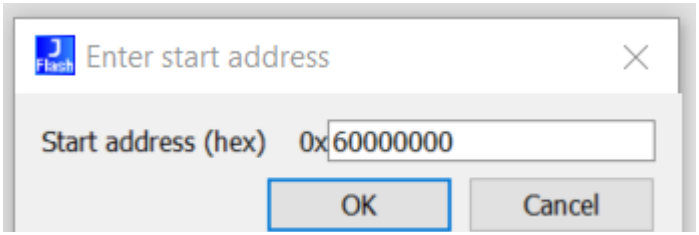


图3

设置Start address为0x60000000(如下图所示)，在与RT1064成功建立连接后，即可将BIN文件烧录到外置QSPI。



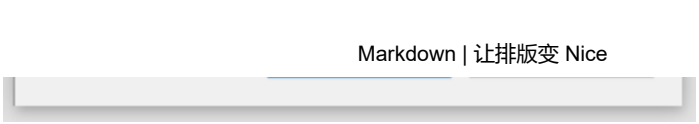


图4

4.烧录fbootloader 代码到内部QSPI

烧录bootloader代码到内部QSPI就比较简单了，与调试SDK library中的其他demo无异，这里小编使用MIMXRT1064-EVK板载的OpenSDA调试接口，在MCUXpresso IDE中单击debug按钮来实现代码烧录。

5. 打开串口终端测试。

在串口终端中配置完成后，连接串口，输入's'，即可完成代码跳转，LED随即周期性闪烁。