M032 ISR In SRAM

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code uses M032 to run ISR in SRAM.
BSP Version	M031_Series_BSP_CMSIS_V3.03.000
Hardware	NuMaker-M032SE V1.3

The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.

Nuvoton is providing this document only for reference purposes of NuMicro microcontroller and microprocessor based system design. Nuvoton assumes no responsibility for errors or omissions.

All data and specifications are subject to change without notice.

For additional information or questions, please contact: Nuvoton Technology Corporation.

www.nuvoton.com

1 Overview

This example code uses the M032 ISP function to map UART Handler to SRAM, so that UART function can work normally while FMC operates Flash memory.

The chips that supports mapping interrupt vector table to SRAM are shown in Table 1-1. The specific settings need to be modified according to different situations.

The series Not supports the function mapped interrupt vector table to SRAM	The series supports the function mapped interrupt vector table to SRAM
M051	All Cortex®-M4 (VTOR Register)
M058S	All Cortex®-M23 (VTOR Register)
M0518	M031 (FMC support)
M0519	M0564 (FMC support)
Mini51	Nano103 (FMC support)
Mini55	NUC121/125 (FMC support)
Mini57	NUC126 (FMC support)
Mini58	NUC029xGE (FMC support)
Nano1x2	
Nano100	
NUC100	
NUC122	
NUC123	
NUC131	
NUC200	
NUC029xAN	
NUC029xAE	
NUC029xDE	
NUC029xEE	

Table 1-1 Supports the Function Mapped Interrupt Vector Table to SRAM

1.1 Principle

When Flash memory controller (FMC) is busy in running program and operating Flash in the same bank, any CPU access to Flash memory will cause CPU halt for waiting FMC ready. If the FMC is busy in page erasing, it may cause CPU halt for a long time to erase pages.

To avoid this situation, user needs to avoid CPU to access Flash memory when page erasing. Or user can copy interrupt handle to SRAM and use ISP function mapping all exceptions to SRAM to execute code.

In IAP mode, ISP function could be used to map 512 bytes SRAM to vector map space. By executing code in SRAM, CPU will not access Flash to get instructions. By mapping all exceptions to SRAM, all interrupts will not need to get exception handler from Flash memory.

Then, if any exceptions assert, CPU can read exception handler from SRAM with zero wait state to speed up exception latency. Figure 1-1 shows the programming flow of remapping interrupt by VECMAP.

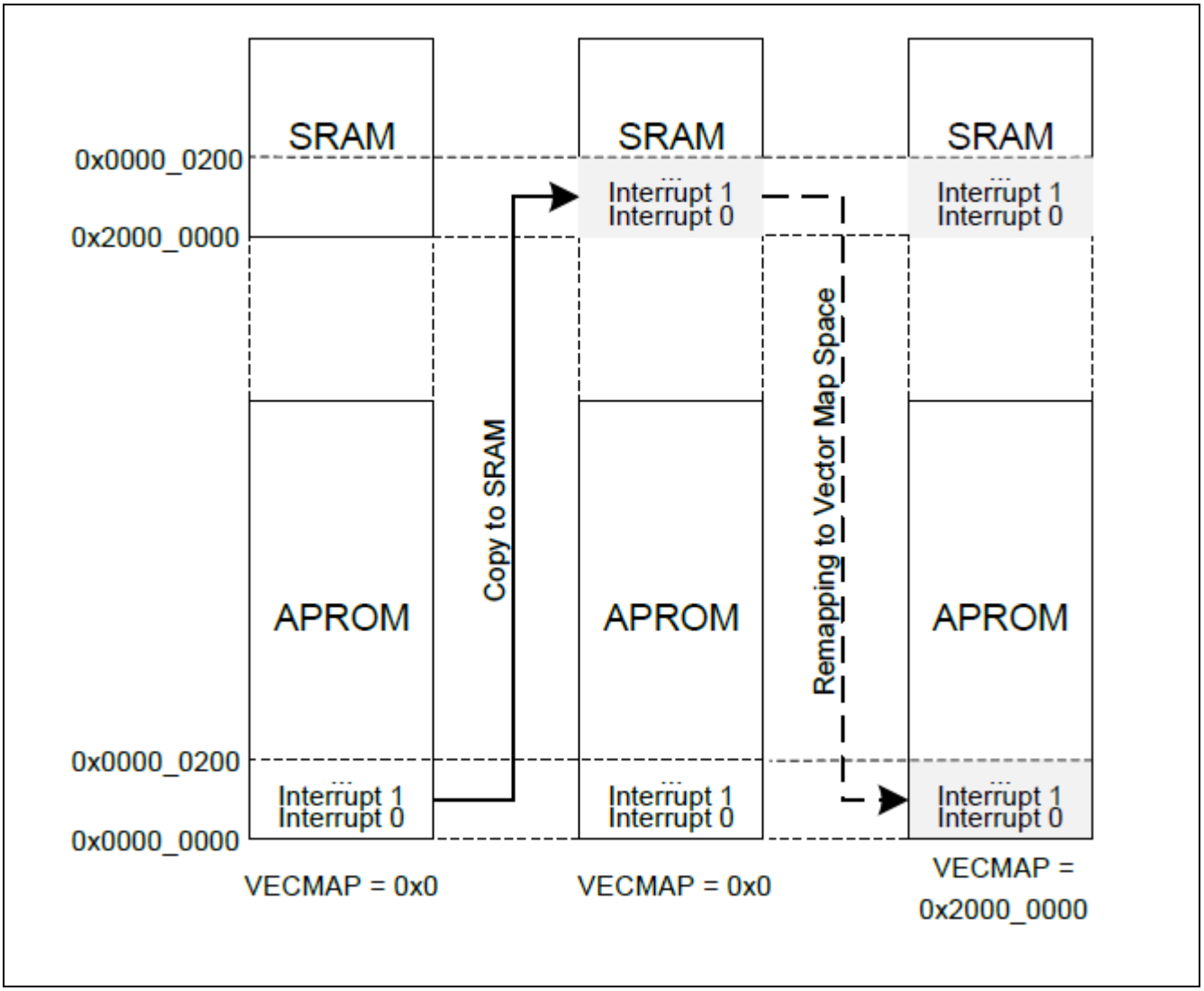

Figure 1-1 Remapping Vector Page to SRAM Using ISP

This function needs to enable the chip IAP function. The chip configuration is shown in Figure 1-2.

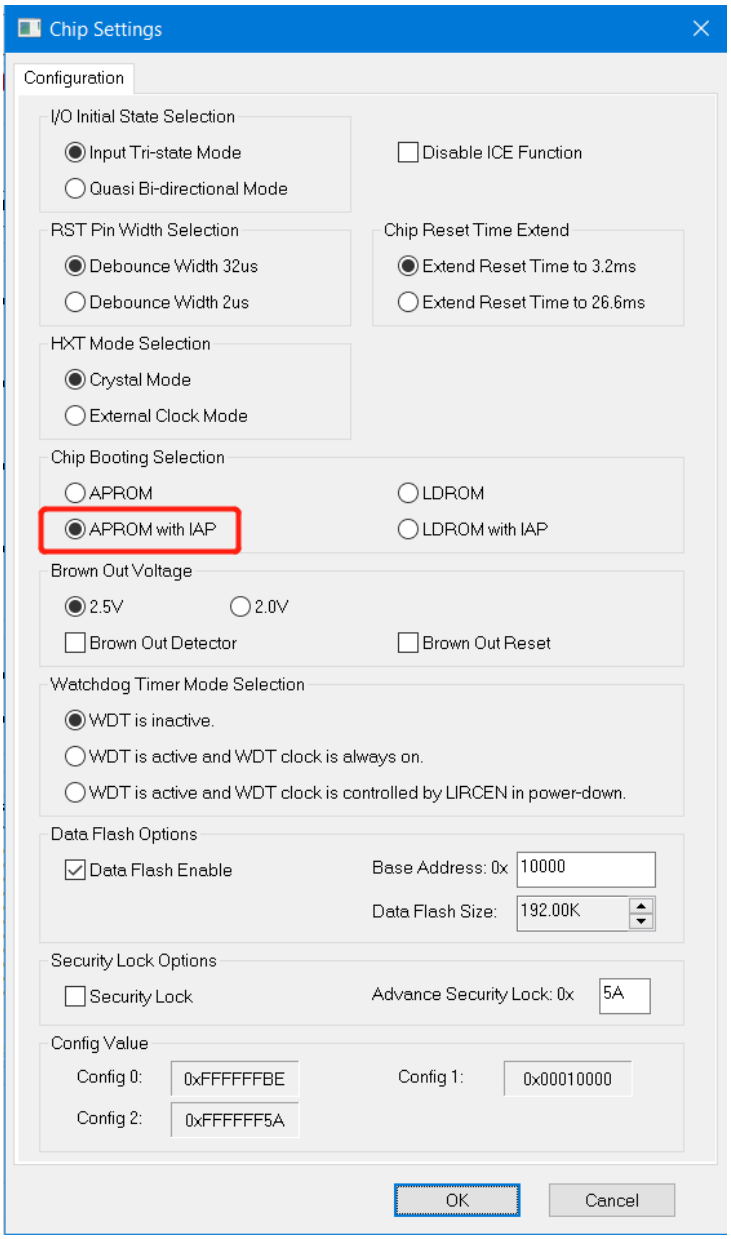

Figure 1-2 Chip Configuration

1.2 Demo Result

1.2.1 System Vector Space Not Mapped to SRAM Memory

Enter Keil, select **Project** -> **Options for Target** 'ISR_In_SRAM', and on "Preprocessor Symbols" keep no REMAP_ISR_IN_SRAM in "Define". Click Build to compile the project. After compiling, download it to the development board.

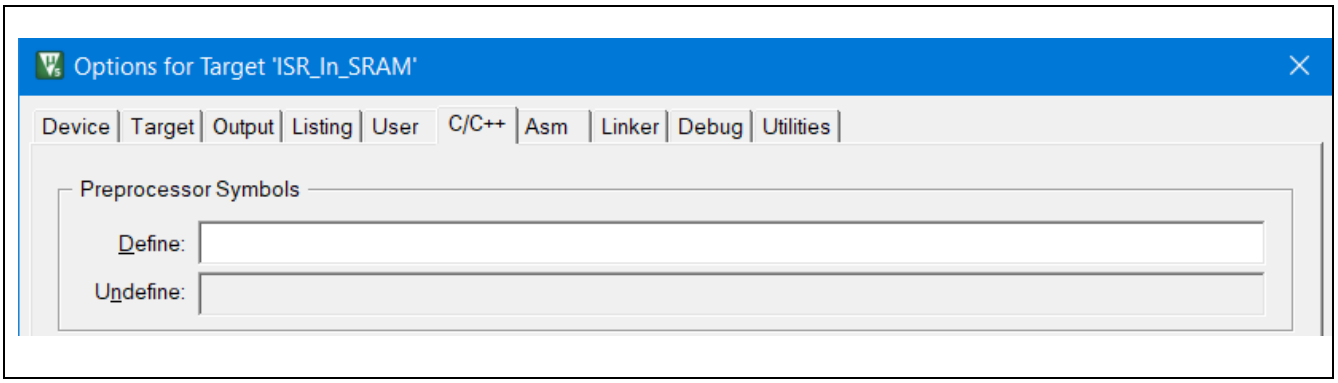

Figure 1-3 Remove REMAP_ISR_IN_SRAM in Define

Open the serial port tool eaglecom and use the " Period " function to send 400 characters '1' every 1ms and send data to the serial port of the development board all the time. The running results are shown in Figure 1-4. A total of 95600 bytes are sent and only 6480 bytes are actually received.

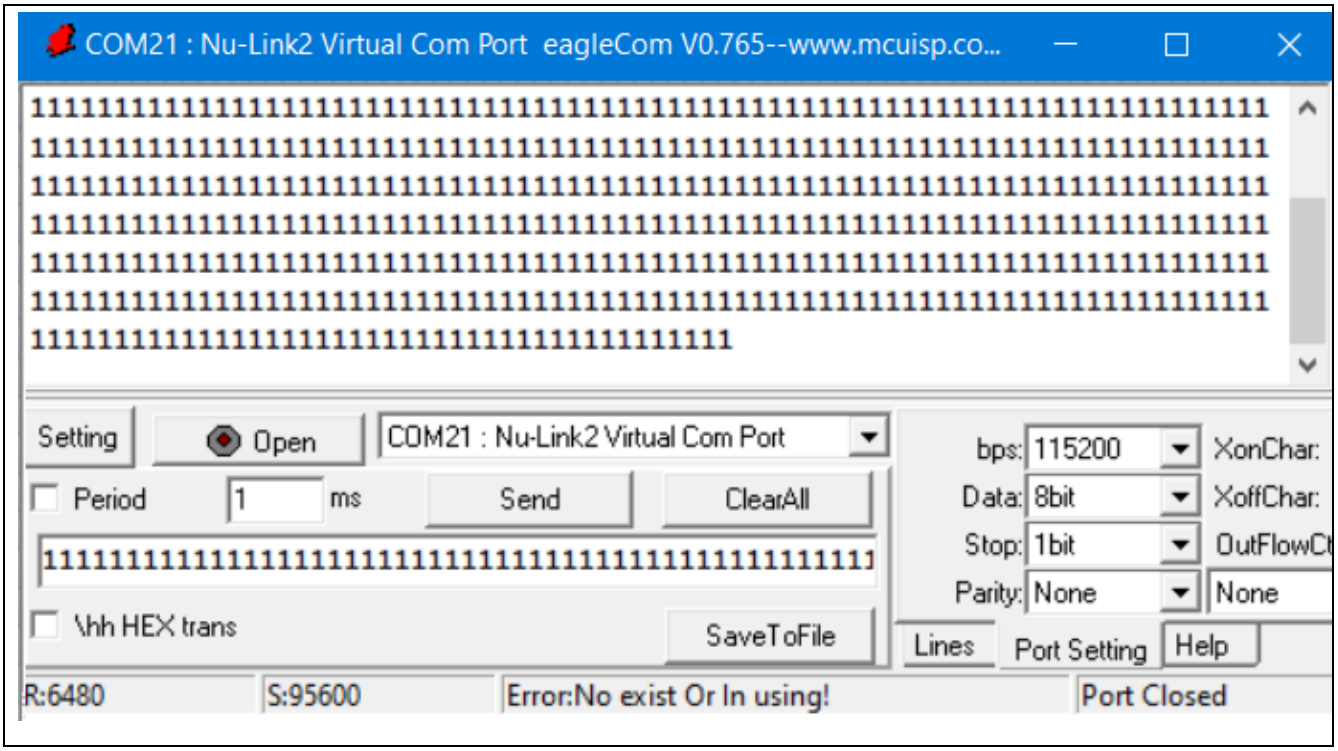

Figure 1-4 Vector Page Mapped on Flash Results

Use the logic analyzer to capture the UART data waveform. It can be seen that M032 has a 19.2ms space time period during receiving the UART data, and does not process and return the UART data. This is because when M032 FMC erases the designated area of Flash, the chip cannot access the programs in the same bank at the same time, so the programs stored in Flash cannot be executed in time.

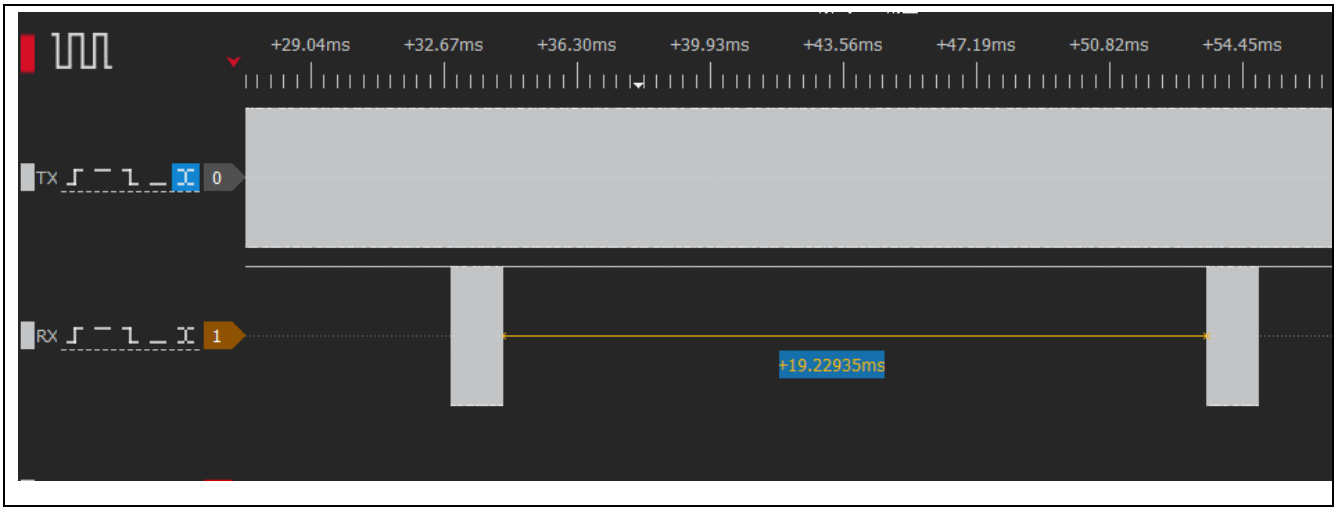

Figure 1-5 When FMC Accesses Flash UART Receives and Sends 1

1.2.2 System Vector Space Mapped to SRAM Memory

Enter Keil, select **Project** -> **Options for Target** 'ISR_In_SRAM', and on "Preprocessor Symbols" keep REMAP_ISR_IN_SRAM in "Define". Click Build to compile the project. After compiling, download it to the development board.

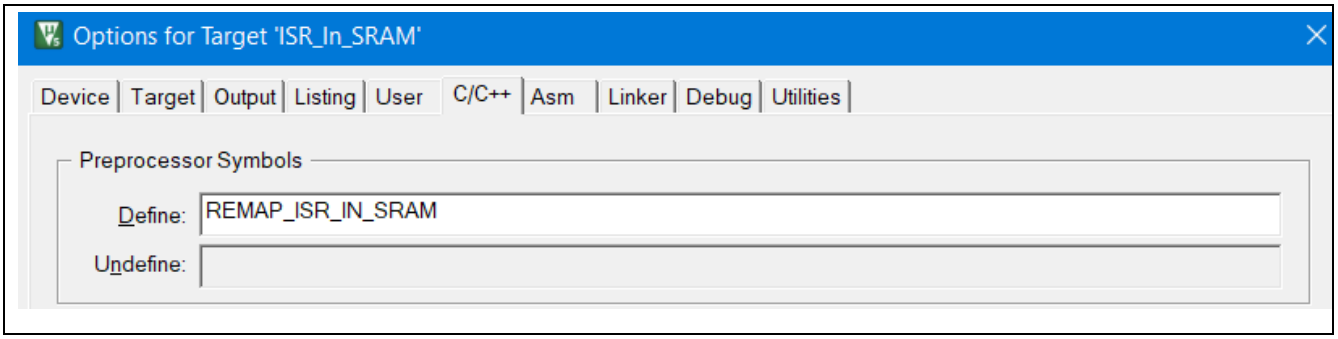

Figure 1-6 Add REMAP_ISR_IN_SRAM in Define

Open the serial port tool eaglecom and use the "Period" function to send 400 characters ’1’ every 1ms and send data to the serial port of the development board all the time. The running results are shown in Figure 1-7. It can be seen that a total of 59200 bytes are sent and 59200 bytes are actually received.

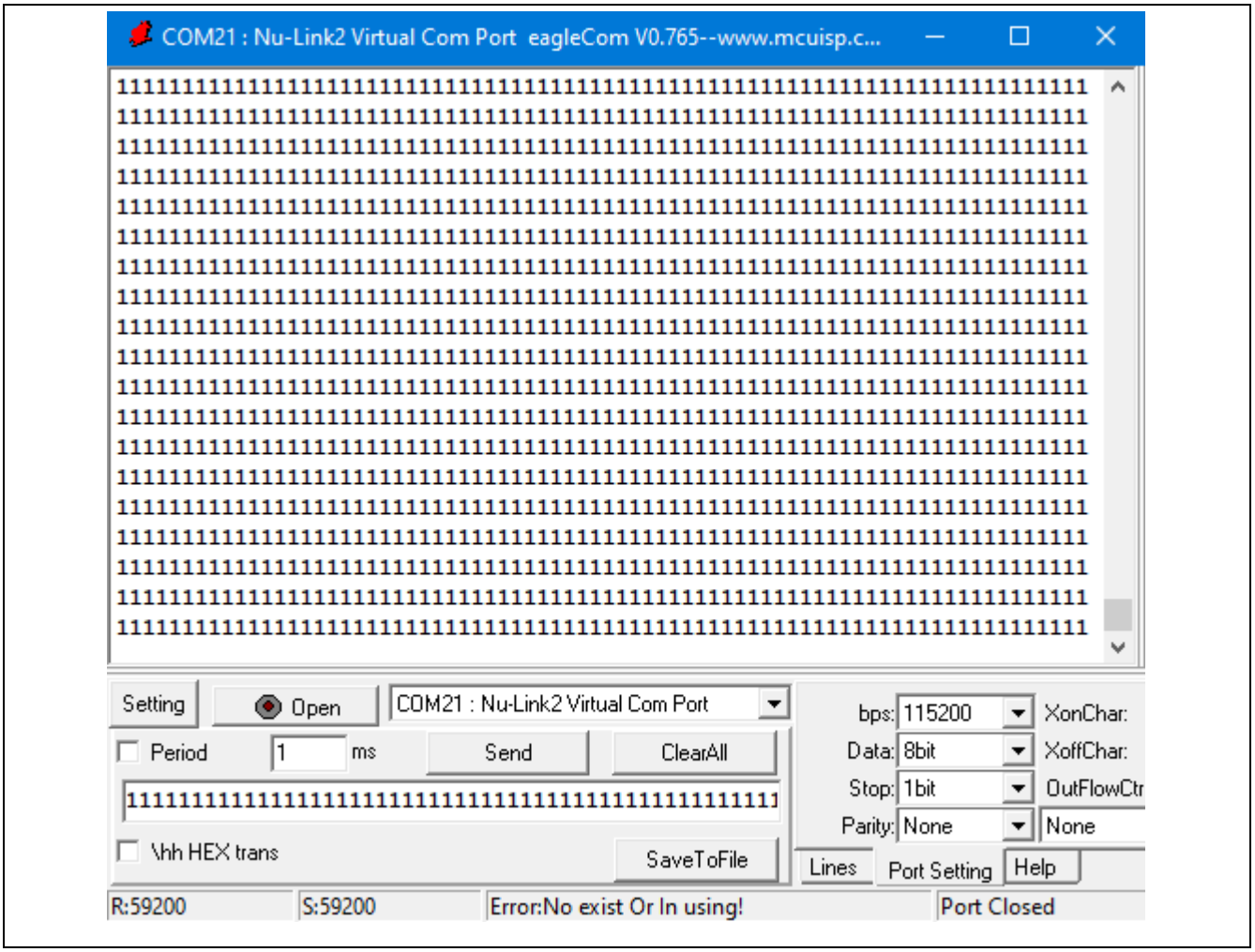

Figure 1-7 Vector Page Remapped on SRAM Results

Use the logic analyzer to capture the data waveform of UART, as shown in Figure 1-8. The M032 can receive and process the UART data normally after receiving it even in FMC is busy in page erasing. All programs are executed in SRAM and the interrupt vector page is remapped in SRAM, which can avoid the CPU's long-time halt and waiting when Flash programming.

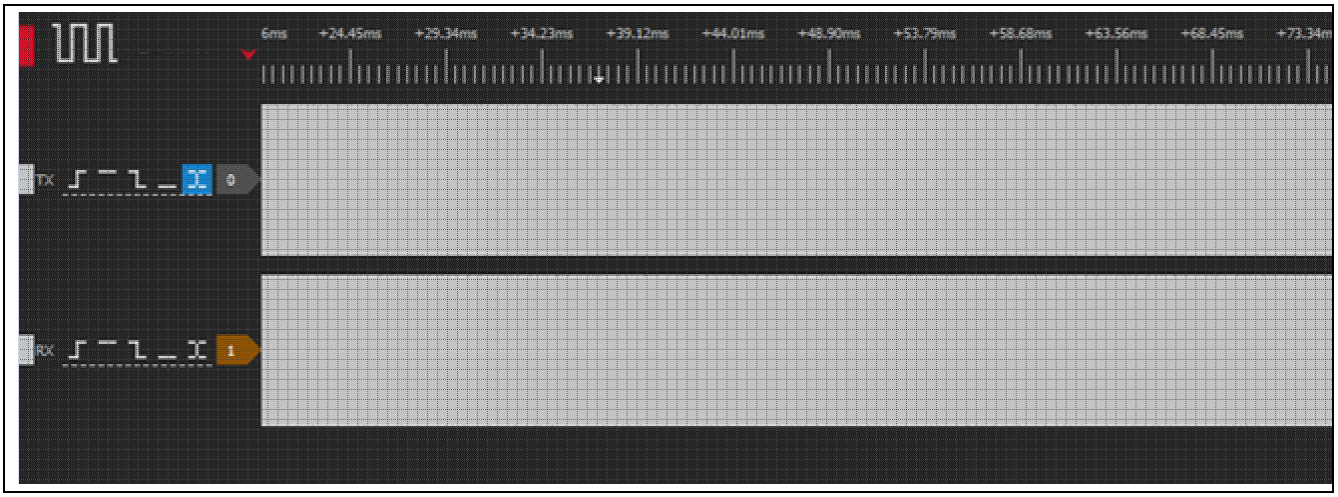

Figure 1-8 When FMC Accesses Flash UART Receives and Sends 2

2 Code Description

Use the scatter file to allocate chip memory storage space, and divide the continuous 512 bytes address space required for interrupt vector space in SRAM.

```
FLASH 0x00000000
{
    ER_STARTU +0
    {
        startup_M031Series_SRAM.o(RESET, +First)
    }
    ER_RO +0
    {
        *(+RO)
    }

    ;on SRAM region
    VECTORS_SRAM 0x20000000 0x200 {
        *(VectorSection)
    }
    FASTCODE_INIT +0
    {
        uart.o(+RO)
        *(fastcode)
    }
    ER_RW +0
    {
       *(+RW, +ZI)
    }
}
```

Customize the code snippet FASTCODE_INIT in SRAM, and put the interrupt handling function and the function to be called in this space. Note that in this example, the erase or write operation of Flash by FMC also needs to be placed in SRAM.

```
#pragma arm section code="fastcode"

void UART02_IRQHandler(void)
{
    uint8_t u8InChar = 0xFF;

    if (UART_GET_INT_FLAG(UART0, UART_INTSTS_RDAINT_Msk))
    {
        /* Get all the input characters */
        while (UART_IS_RX_READY(UART0))
        {
            /* Get the character from UART Buffer */
            u8InChar = UART_READ(UART0);

            UART_WRITE(UART0, u8InChar);
        }
    }

    if (UART0->FIFOSTS & (UART_FIFOSTS_BIF_Msk | UART_FIFOSTS_FEF_Msk |
UART_FIFOSTS_PEF_Msk | UART_FIFOSTS_RXOVIF_Msk))
    {
        UART_ClearIntFlag(UART0, (UART_INTSTS_RLSINT_Msk | UART_INTSTS_BUFERRINT_Msk));
    }
```

```
}

void FMC_ISPGO(void)
{
    FMC->ISPTRG = FMC_ISPTRG_ISPGO_Msk;

    while (FMC->ISPTRG & FMC_ISPTRG_ISPGO_Msk) { }
}

#pragma arm section
```

Copy the interrupt vector page to the SRAM specified address and set the interrupt vector remapping address.

```
#define VECMAP_SRAM_BASE    0x20000000UL
uint8_t u8Vectors_SRAM[512] __attribute__((section("VectorSection")));

    /* Lock protected registers */
    SYS_LockReg();

    if (FMC_GetVECMAP() != VECMAP_SRAM_BASE)
    {
        printf("Before setting VECMAP=0x%x\n\r", FMC_GetVECMAP());

        /* Mask all interrupt before changing VECMAP to avoid wrong interrupt handler
fetched */
        __set_PRIMASK(1);

        memcpy(u8Vectors_SRAM, &__Vectors, 512);

        /* Unlock protected register */
        SYS_UnlockReg();

        /* Enable FMC ISP function */
        FMC_Open();

        /* set VECMAP to SRAM */
        FMC_SetVectorPageAddr((uint32_t)&u8Vectors_SRAM);

        /* Disable FMC ISP function */
        FMC_Close();

        /* Lock protected Register */
        SYS_LockReg();

        __set_PRIMASK(0);
```

```
        printf("After  setting VECMAP=0x%x\n\r", FMC_GetVECMAP());
    }
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M031_Series_BSP_CMSIS_V3.03.000
- IDE version
  - Keil uVersion 5.35

3.2 Hardware Requirements

- Circuit components
  - NuMaker-M032SE V1.3

4 Directory Information

The directory structure is shown below.

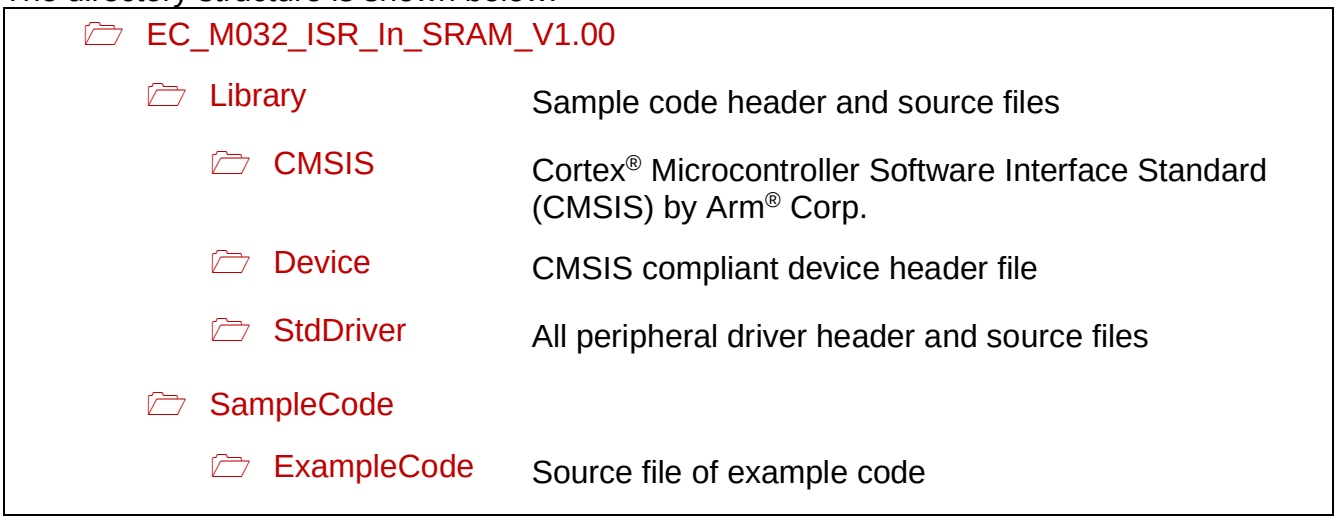

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *ISR_In_SRAM.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
3. Open the serial port tool and select the corresponding serial port.
   - Press the reset key and view the serial port debug information
   - Use the logic analyzer to capture serial communication waveform

6 Revision History

Date	Revision	Description
2021.07.16	1.00	1. Initially issued.

Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

Please note that all data and specifications are subject to change without notice.
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.