M480 UART RX PDMA

Example Code Introduction for 32-bit NuMicro® Family

Information

Application	This example code is used for the M480 series microcontroller (MCU) to receive UART data through PDMA.
BSP Version	M480 BSP CMSIS V3.05.001
Hardware	NuMaker-PFM-M487 Ver3.0

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

www.nuvoton.com

1 Overview

This example code is used to configure UART and PDMA, then transfer data from UART_RX to RAM through PDMA, and also provide complete package flag. Through Scatter-Gather operation mode, the example code will transfer data by looped around two descriptor tables from UART RX to two different destination RAMs in sequence. Complete package flag is used to indicate that the transmission is completed or timeout. This document introduces the example code of UART_RX triggering PDMA configuration, PDMA Scatter-gather mode configuration, descriptor tables configuration, complete package flag and so on.

1.1 Demo Result

If UART1 and debug UART are connected to PC when executing the program, one UART tool can send data to M480 and the other UART tool can display data from M480 as shown below.

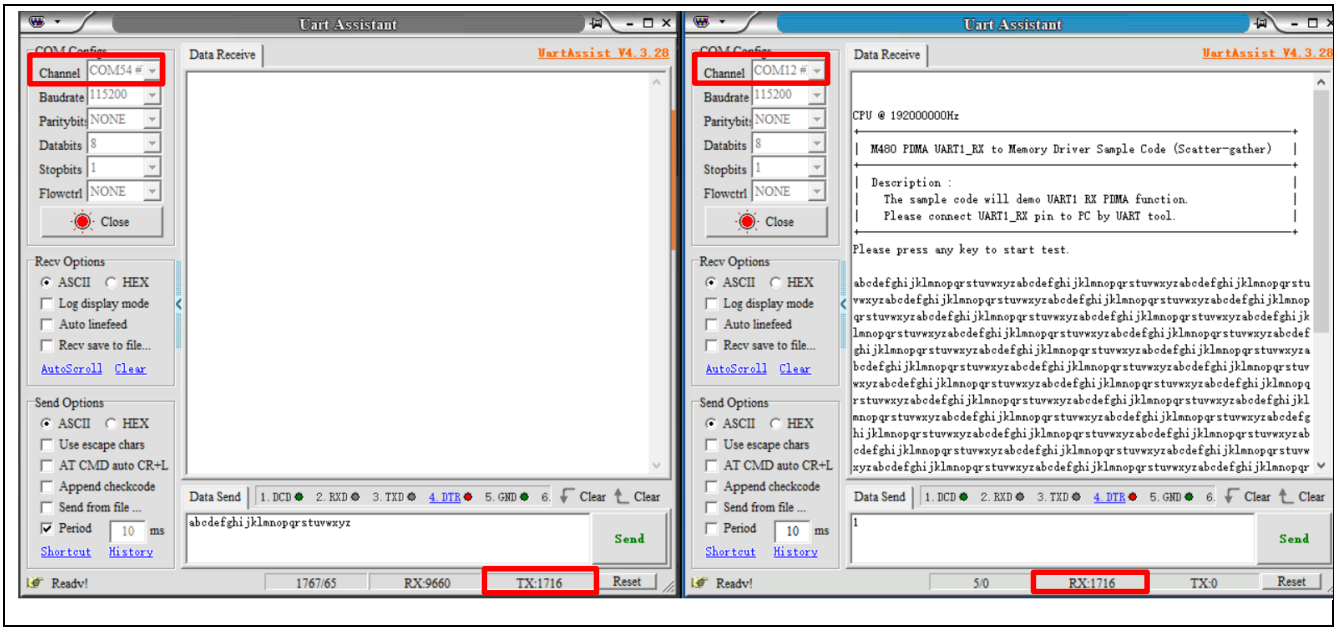

Figure 1-1 Demo Result

2 Code Description

Define PDMA channel:

```
#define UART_RX_DMA_CH 0
```

Define UART RX buffer size:

```
#define RXBUFSIZE 1024
```

Define PDMA descriptor tables:

```
typedef struct dma_desc_t
{
    uint32_t ctl;
    uint32_t src;
    uint32_t dest;
    uint32_t offset;
} DMA_DESC_T;
DMA_DESC_T DMA_DESC[2];
```

PDMA initialization:

```
void PDMA_Init(void)
{
    /* Open Channel 1 */
    PDMA_Open(PDMA, 1 << UART_RX_DMA_CH);
    /* Enable Scatter Gather mode, assign the first scatter-gather descriptor table is
table 1, and set transfer mode as uart rx to memory */
  PDMA_SetTransferMode(PDMA, UART_RX_DMA_CH, PDMA_UART1_RX, TRUE, (uint32_t)&DMA_DESC[0]);


    /* Scatter-Gather descriptor table configuration in SRAM */
    /*-------------------------------------------------------------------------------------
      Descriptor table 1 configuration:
                 UART1_BASE                          g_u8RecData[0]
               ---------------------------  -->  ---------------------------
             /| [0]  | [1]  |  [2] |  [3] |     | [0]  | [1]  |  [2] |  [3] |\
              |      |      |      |      |     |      |      |      |      |
  RXBUFSIZE/2 |            ...            |     |            ...            | RXBUFSIZE/2
              |      |      |      |      |     |      |      |      |      |
             \| [508]| [509]| [510]| [511]|     | [508]| [509]| [510]| [511]|/
               ---------------------------       ---------------------------
               \                         /       \                         /
                     8bits(one byte)                   8bits(one byte)


        Operation mode = scatter-gather mode
```

```
        Next descriptor table = DMA_DESC[1](Descriptor table 2)
        transfer done and timeout interrupt = enable

        Transfer count = RXBUFSIZE/2
        Transfer width = 8 bits
        Source address = UART1_BASE
        Source address increment size = fixed address(no increment)
        Destination address = g_u8RecData
        Destination address increment size = 8 bits(one byte)
        Transfer type = single transfer

        Total transfer length = (RXBUFSIZE/2) * 8 bits
    ------------------------------------------------------------------------------------*/
    DMA_DESC[0].ctl = \
  ((RXBUFSIZE / 2 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is RXBUFSIZE/2 */ \
   PDMA_WIDTH_8 |    /* Transfer width is 8 bits */ \
   PDMA_SAR_FIX |    /* Source increment size is  fixed(no increment) */ \
   PDMA_DAR_INC |    /* Destination increment size is 8 bits */ \
   PDMA_REQ_SINGLE | /* Transfer type is single transfer type */ \
   PDMA_BURST_1 |    /* Burst size is 1. No effect in single transfer type */ \
   PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */
    /* Configure source address */
    DMA_DESC[0].src = (uint32_t)UART1_BASE;
    /* Configure destination address */
    DMA_DESC[0].dest = (uint32_t)&g_u8RecData[0];/*buffer 1 */
    /* Configure next descriptor table address */
    DMA_DESC[0].offset = (uint32_t)&DMA_DESC[1] - (PDMA->SCATBA); /* next operation table
is table 2 */


    /*------------------------------------------------------------------------------------
      Descriptor table 2 configuration:
                UART1_BASE                          g_u8RecData[RXBUFSIZE/2]
              ---------------------------  -->  ---------------------------
            /| [0]  | [1]  |  [2] |  [3] |     | [0]  | [1]  |  [2] |  [3] |\
             |     |      |     |      |     |     |      |     |      |     |
  RXBUFSIZE/2 |           ...           |     |           ...           | RXBUFSIZE/2
             |     |      |     |      |     |     |      |     |      |     |
            \| [508]| [509]| [510]| [511]|     | [508]| [509]| [510]| [511]|/
              ---------------------------       ---------------------------
              \                        /       \                         /
                    8bits(one byte)                   8bits(one byte)
```

```
        Operation mode = scatter-gather mode
        Next descriptor table = DMA_DESC[0](Descriptor table 1)
        transfer done and timeout interrupt = enable

        Transfer count = RXBUFSIZE/2
        Transfer width = 8 bits
        Source address = UART1_BASE
        Source address increment size = fixed address(no increment)
        Destination address = g_u8RecData[RXBUFSIZE/2]
        Destination address increment size = 8 bits(one byte)
        Transfer type = single transfer

        Total transfer length = (RXBUFSIZE/2) * 8 bits
    ------------------------------------------------------------------------------------*/
    DMA_DESC[1].ctl = \
  ((RXBUFSIZE / 2 - 1) << PDMA_DSCT_CTL_TXCNT_Pos) | /* Transfer count is RXBUFSIZE/2 */ \
   PDMA_WIDTH_8 |    /* Transfer width is 8 bits */ \
   PDMA_SAR_FIX |    /* Source increment size is fixed(no increment) */ \
   PDMA_DAR_INC |    /* Destination increment size is 8 bits */ \
   PDMA_REQ_SINGLE | /* Transfer type is single transfer type */ \
   PDMA_BURST_1 |    /* Burst size is 1. No effect in single transfer type */ \
   PDMA_OP_SCATTER;  /* Operation mode is scatter-gather mode */
    /* Configure source address */
    DMA_DESC[1].src = (uint32_t)UART1_BASE;
    /* Configure destination address */
    DMA_DESC[1].dest = (uint32_t)&g_u8RecData[RXBUFSIZE / 2] ; /* buffer 2 */
    /* Configure next descriptor table address */
    DMA_DESC[1].offset = (uint32_t)&DMA_DESC[0] - (PDMA->SCATBA); /* next operation table
is table 1 */


    /* PDMA channel  timeout*/
    PDMA_SetTimeOut(PDMA, UART_RX_DMA_CH, 1, PDMA_TIME); /* channel 0 timeout period = CLK
* TOC0  */


    /* Enable interrupt */
    PDMA_EnableInt(PDMA, UART_RX_DMA_CH, PDMA_INT_TRANS_DONE); /* channel transfer done */
    PDMA_EnableInt(PDMA, UART_RX_DMA_CH, PDMA_INT_TIMEOUT); /*channel timeout */
    NVIC_EnableIRQ(PDMA_IRQn);
}
```

UART initialization:

```
        /* Init UART1 */
```

```
        UART1_Init();
```

Data processing:

```
    while (1)
    {
        /* Add data processing code */
        if (g_u32comRbytes)
        {
            uint16_t tmp;
            tmp = g_u32comRtail;

            if (g_u32comRhead != tmp)
            {
                printf("%c", g_u8RecData[g_u32comRhead]);
                g_u32comRhead = (g_u32comRhead == (RXBUFSIZE - 1)) ? 0 : (g_u32comRhead +
1);
                g_u32comRbytes--;
            }
        }
    }
```

Update UART RX buffer:

```
void PDMA_IRQHandler(void)
{
    uint32_t status = PDMA_GET_INT_STATUS(PDMA); /* interrupt status */
    uint32_t cnt = PDMA_GET_TRANS_CNT(PDMA, UART_RX_DMA_CH); /* channel transfer count */

    if (status & 0x2)     /* channel transfer done */
    {
        /* Check channel transfer done status */
        if (PDMA_GET_TD_STS(PDMA) == PDMA_TDSTS_TDIF0_Msk) /* finish a descriptor table*/
        {
            /* Update receive count */
            g_u32comRbytes += g_u32cntbak + 1; /* count */
            g_u32comRtail = (g_u32comRtail + g_u32cntbak + 1) % RXBUFSIZE; /* index */
            g_u32cntbak = RXBUFSIZE / 2 - 1; /* channel cnt backup */

            g_u8PackageComplete = 1;

            /* Clear transfer done flag of channel UART_RX_DMA_CH */
            PDMA_CLR_TD_FLAG(PDMA, PDMA_TDSTS_TDIF0_Msk);
        }
    }
```

```
    else if (status & 0x100)     /* channel 0 timeout */
    {
        /* Update receive count */
        g_u32comRbytes += g_u32cntbak - cnt;  /* count */
        g_u32comRtail = (g_u32comRtail + g_u32cntbak - cnt) % RXBUFSIZE;  /* index */
        g_u32cntbak = cnt; /* channel cnt backup */

        g_u8PackageComplete = 0;

        /* restart timeout */
        PDMA_SetTimeOut(PDMA, UART_RX_DMA_CH, 0, 0);
        PDMA_CLR_TMOUT_FLAG(PDMA, UART_RX_DMA_CH);
        PDMA_SetTimeOut(PDMA, UART_RX_DMA_CH, 1, PDMA_TIME);
    }
    else
        printf("unknown interrupt !!\n");
}
```

3 Software and Hardware Requirements

3.1 Software Requirements

- BSP version
  - M480 Series BSP CMSIS v3.05.001
- IDE version
  - Keil uVersion 5.26

3.2 Hardware Requirements

- Circuit components
  - NuMaker-PFM-M487 Ver3.0
  - PC
  - Nu-Bridge

- Pin Connect

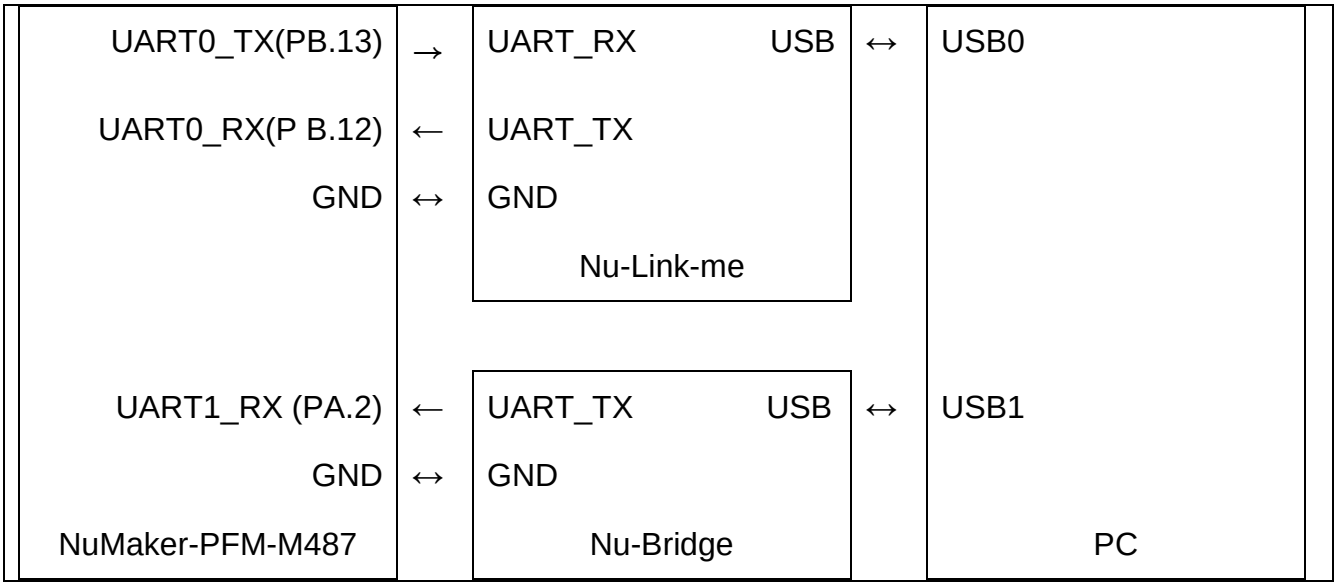

Figure 3-1 Pin Connect

4 Directory Information

The directory structure is shown below.

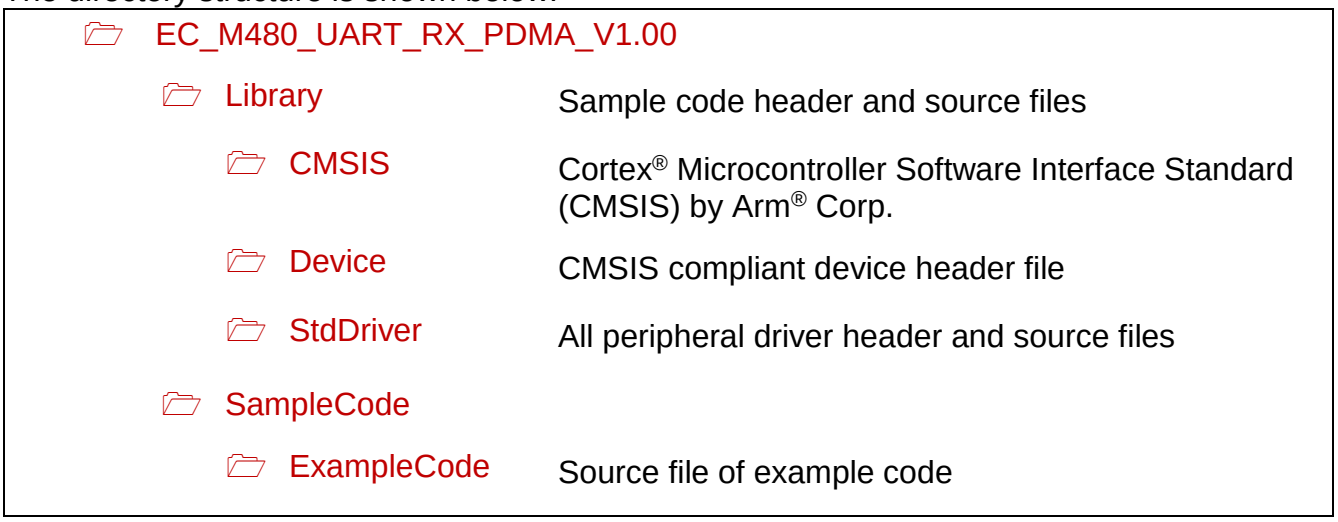

Figure 4-1 Directory Structure

5 Example Code Execution

1. Browse the sample code folder as described in the Directory Information section and double-click *UART_RX_PDMA_ScatterGather.uvproj*.
2. Enter Keil compile mode.
   - Build
   - Download
   - Start/Stop debug session
3. Enter debug mode.
   - Run
4. UART receives data.
   - Open *UartAssist.exe*
   - Baud rate: 115200bps
5. Print debug information.
   - Open *UartAssist.exe*
   - Baud rate: 115200bps

6 Revision History

Date	Revision	Description
2021.08.13	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.