Application Note

32-bit Cortex™-M0 MCU
NuMicro® Family

How to use PDMA to transfer data in NUC1xx?

Application Note

Table of Contents-

1 INTRODUCTION

This document explains the sample code, "Smpl_DrvPDMA" which is included in the ***AN_1016_EN.ZIP*** file and demonstrates how to transfer data between RAM and UART receive/transmit register by PDMA.

1.1 Scope

This article is provided for programmers applying PDMA into applications to improve the performance of system.

1.2 Basic acknowledge

- What is NUC1xx's PDMA?

  NUC1xx's PDMA is the peripheral direct memory access controller that transfers data to and from memory or transfer data to and from APB(Advanced Peripheral Bus). It has nice channels of DMA associated with Peripheral-to-Memory, Memory-to-Peripheral or Memory-to-Memory.

1.3 Features

- Illustrates how to transfer data from memory to peripheral device.
- Illustrates how to transfer data from peripheral device to memory.
- Interrupt triggered after transfer is finished.

1.4 Limitation

- Only illustrates transfer data between memory and peripheral device with 8 bits width data. (It's easily to change parameters to transfer between memory and memory, or changed to other data width).

2 CODE SECTION

2.1 Globals and Macros

Global SrcArray is defined to fill the pre-defined data that will transfer to UART Transmit Holding Register (UA_THR).

Global DestArray is defined to receive data from UART Receive Buffer Register (UA_RBR).

IntCnt will record the PDMA transfer count.

IsTestOver is the flag set by PDMA interrupt after all data transfer is finished.

```
#define      UART_TEST_LENGTH                  64
uint8_t SrcArray[UART_TEST_LENGTH];
uint8_t DestArray[UART_TEST_LENGTH];
int32_t IntCnt;
volatile int32_t IsTestOver;
```

2.2 Main Function (in Smpl_DrvPDMA.c)

(Please skip this section if you just want to know the flow of PDMA transfer)

The main function sets the system clock's frequency by assign value to "SystemFrequecy", and then opens two UART port by function "DrvUART_Open()", and finally it call PDMA_UART() to begin PDMA flow.

In the flowing code, 1st UART port (UART_PORT0) will be used to output/input messages in console, 2nd UART port (UART_PORT1) will be used to demonstrate the PDMA transfer between memory and peripheral device.

Code for open two UART:

```
/* Set UART Pin */
DrvGPIO_InitFunction(FUNC_UART0);
DrvGPIO_InitFunction(FUNC_UART1);

/* UART Setting */
sParam.u32BaudRate        = 115200;
sParam.u8cDataBits        = DRVUART_DATABITS_8;
sParam.u8cStopBits        = DRVUART_STOPBITS_1;
sParam.u8cParity          = DRVUART_PARITY_NONE;
sParam.u8cRxTriggerLevel  = DRVUART_FIFO_1BYTES;

/* Set UART Configuration */
DrvUART_Open(UART_PORT0,&sParam);
DrvUART_Open(UART_PORT1,&sParam);
```

2.3 The PDMA transfer flow (in Smpl_DrvPDMA.c)

2.3.1 Transfer flow in process

Function PDMA_UART() demonstrates the whole procedure of PDMA transfer.

Before transfer start, it prepared the transfer data and initialize PDMA module firstly.

1. Calls BuildSrcPattern() to fill the source data in global SrcArray;
2. Calls DrvPDMA_Init() to initialize the PDMA module;
3. Calls DrvPDMA_SetCHForAPBDevice() with parameter eDRVPDMA_UART1 and eDRVPDMA_UART0, so as to associate PDMA with device UART1/UART0.

```
BuildSrcPattern((uint32_t)SrcArray,UART_TEST_LENGTH);


UARTPort = UART1_BASE;
i=UART_TEST_LENGTH;
ClearBuf((uint32_t)DestArray, UART_TEST_LENGTH,0xFF);


/* PDMA Init */
DrvPDMA_Init();


/* PDMA Setting */
DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_1,eDRVPDMA_UART1,eDRVPDMA_WRITE_APB);
DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_0,eDRVPDMA_UART1,eDRVPDMA_READ_APB);
```

After the initialization above finished, it sets the two transfer channels for write and read data respectively, the transfer address and data width also specified in this step.

(Please note that the UART1 transmit and receive register has the same address, which is defined as UARTPort here.)

```
/* CH1 TX Setting */
sPDMA.sSrcAddr.u32Addr          = (uint32_t)SrcArray;
sPDMA.sDestAddr.u32Addr         = UARTPort;
sPDMA.u8TransWidth              = eDRVPDMA_WIDTH_8BITS;
sPDMA.u8Mode                    = eDRVPDMA_MODE_MEM2APB;
sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_INCREMENTED;
sPDMA.sDestAddr.eAddrDirection  = eDRVPDMA_DIRECTION_FIXED;
sPDMA.i32ByteCnt                = UART_TEST_LENGTH;
DrvPDMA_Open(eDRVPDMA_CHANNEL_1,&sPDMA);


/* CH0 RX Setting */
```

```
sPDMA.sSrcAddr.u32Addr          = UARTPort;
sPDMA.sDestAddr.u32Addr         = (uint32_t)DestArray;
sPDMA.u8Mode                    = eDRVPDMA_MODE_APB2MEM;
sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_FIXED;
sPDMA.sDestAddr.eAddrDirection  = eDRVPDMA_DIRECTION_INCREMENTED;
DrvPDMA_Open(eDRVPDMA_CHANNEL_0,&sPDMA);
```

And then install the interrupt handler for PDMA transfer.

```
/* Enable INT */
DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD );
DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD );


/* Install Callback function */
DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_0,eDRVPDMA_BLKD,
  (PFN_DRVPDMA_CALLBACK) PDMA0_Callback );
```

Now start the transfer here. For UART device, it is required to enable PDMA mode by DrvUART_SetPDMA(). For Memory-to-Memory PDMA, do not call DrvUART_SetPDMA().

```
/* Enable UART PDMA and Trigger PDMA specified Channel */
DrvUART_SetPDMA(UART_PORT1,ENABLE);


DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_1);
```

After PDMA transfer is finished, interrupt handler function PDMA0_Callback() will be called to set flag. PDMA_UART() check this flag and then returns.

2.3.2 Transfer flow in interrupt handler

The interrupt handler PDMA0_Callback() will be triggered after one transfer on PDMA channel0 is finished. As the code below, it enables the PDMA transfer again. After triggered for 10 times, it sets the flag IsTestOver to TRUE to indicate the process that all transfer is over.

```
extern int32_t IntCnt;
printf("\tTransfer Done %02d!\r",++IntCnt);


if(IntCnt<10)
{
      DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_1);
      DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
```

```
}
else
{
      IsTestOver = TRUE;
}
```

3 CALLING SEQUENCE

3.1 Run in process

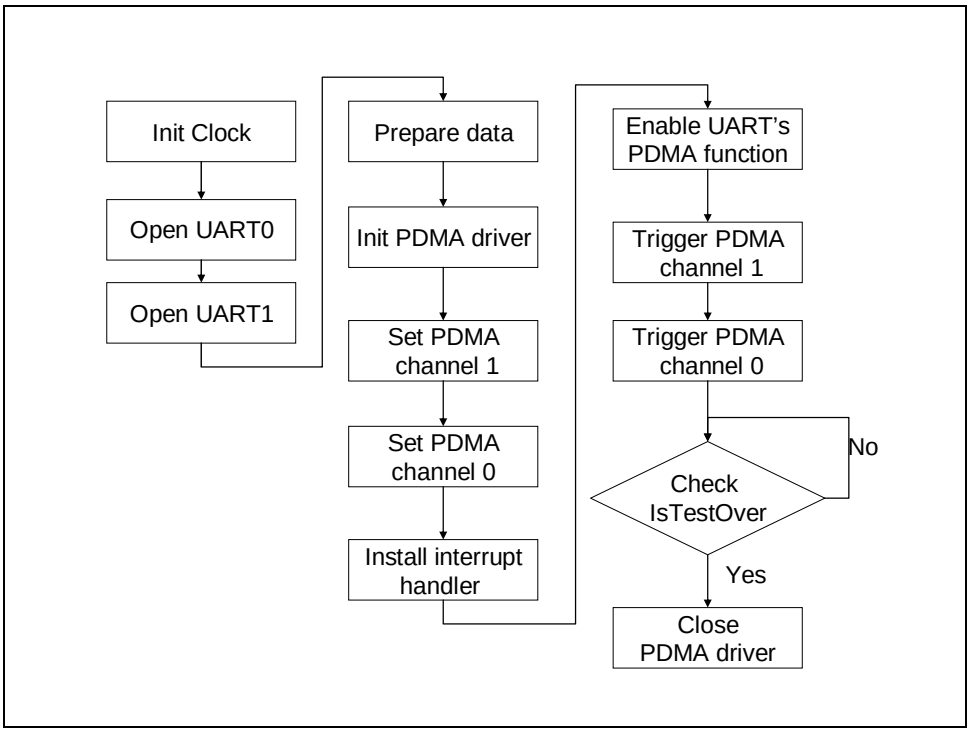

Figure 3-1 Control Flow of Running in Process

1. Initialize Clock Control settings (PLLCON and CLKSEL0 registers)
2. Open UART device for UART0 and UART1
3. Prepare transfer source data.
4. Initialize PDMA driver.
5. Set PDMA channel 1 for memory to UART1
6. Set PDMA channel 0 for UART1 to memory
7. Enable and install PDMA interrupt for PDMA channel 0.
8. Enable UART's PDMA function
9. Trigger PDMA channel 1 to start transfer data to UART1.

10. Trigger PDMA channel 0 to start receiving data from UART1
11. Wait interrupt for setting the flag IsTestOver.
12. Close PDMA driver.

3.2 Run in interrupt handler

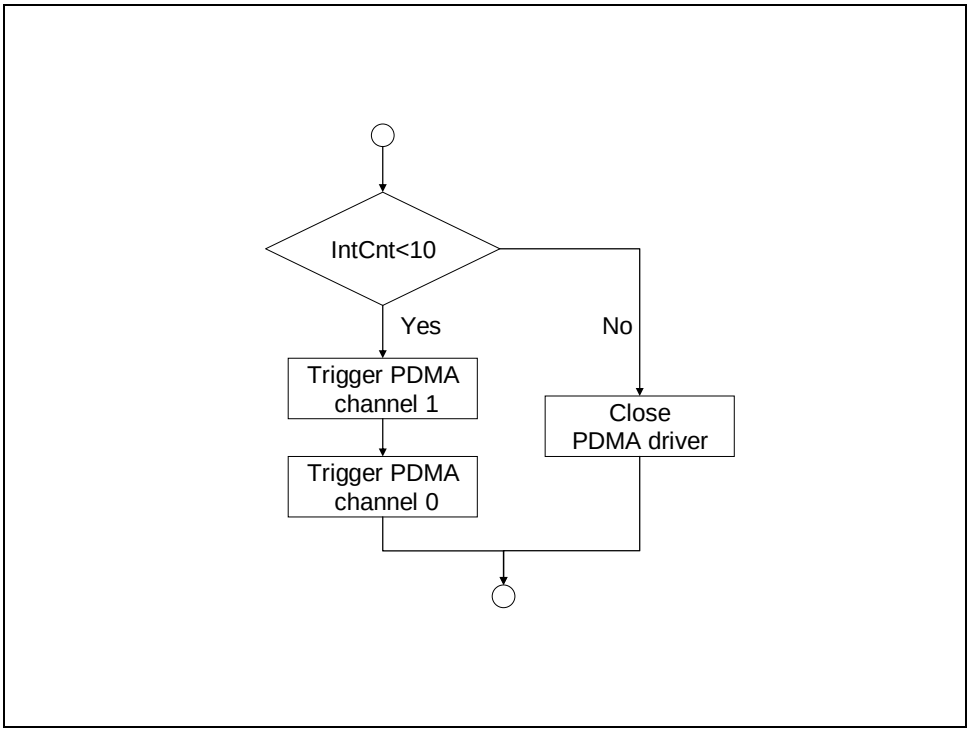

Figure 3-2 Control Flow of Running in Interrupt Handler

1. If the transfer count is smaller than 10
   a) Trigger PDMA channel 1 to start transfer data to UART1.
   b) Trigger PDMA channel 0 to start receiving data from UART1
   c) Increase the transfer count variable(IntCnt).
2. Otherwise (If the transfer count is 10)
   a) Set flag IsTestOver to TRUE.

3.3 API Usage Reference

- NUC1xx Driver Reference Guide.doc

4 EXECUTION ENVIRONMENT SETUP AND RESULT

4.1 Test Smpl_DrvPDMA

The PDMA sample code, Smpl_DrvPDMA, could be built by Keil MDK tool and download to NUC1xx series DEV Board through ICE. Then user is able to execute the code in ICE environment or reset the DEV board to execute the code which had been programmed in on-chip Program Flash to verify the Smpl_DrvPDMA code.

4.2 Result

When running, the source data(SrcArray) will be transferred to UART1 Transmit Holding Register, and data in UART1 Receive Buffer Register will be transferred to the destination buffer (DestArray), so that the data will be copied from SrcArray to DestArray in this way. User can check the data in DestArray after transfer finished to verify the PDMA result.

REVISION HISTORY

REV.	DATE	DESCRIPTION
0.01	March 11, 2010	1. Initially issued.

Important Notice

Nuvoton products are not designed, intended, authorized or warranted for use as components in systems or equipment intended for surgical implantation, atomic energy control instruments, airplane or spaceship instruments, transportation instruments, traffic signal instruments, combustion control instruments, or for other applications intended to support or sustain life. Further more, Nuvoton products are not intended for applications wherein failure of Nuvoton products could result or lead to a situation wherein personal injury, death or severe property or environmental damage could occur.

Nuvoton customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify Nuvoton for any damages resulting from such improper use or sales.

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.