Application Note

32-bit Cortex™-M0 MCU
NuMicro® Family

*在 **NUC1xx** 中如何用PDMA传输数据?*

目录

1 简介

本文利用***AN_1016_EN.ZIP***中的示例程序 “Smpl_DrvPDMA” 项目，阐述了用PDMA在内存和UART串行口之间传输数据的办法。

1.1 目标读者

需要在程序里用PDMA传输数据，以便提高系统效能的读者，建议阅读本文。

1.2 背景知识

- NUC1xx’s 的 PDMA 是什么?

  NUC1xx’s PDMA 是用于直接内存访问的控制器，它可以用来在内存之间或内存/APB(Advanced Peripheral Bus)之间传输数据。该控制器一共有9路DMA, 工作方式可以设定为设备到内存、内存到设备或内存到内存。

1.3 本文涵盖目标

- 阐述了如何从内存到设备传输数据
- 阐述了如何从设备到内存传输数据
- 支持传输完触发中断

1.4 限制

- 仅阐述了内存/设备之间的8位数据宽度的数据传输。(只要修改参数，非常容易改成内存之间传输数据，改用其它的数据宽度也很容易。)

2 代码片断

2.1 全局变量和宏定义

全局变量SrcArray用来存放预定要发送的数据，这些数据将传输到UART Transmit Holding (UA_THR) 寄存器。

全局变量DestArray用来存放接受到的数据，这些数据将从UART Receive Buffer (UA_RBR) 寄存器中接收。

IntCnt 用来记录PDMA 传输次数

IsTestOver 是一个标记， PDMA中断用来通知进程传输已经完成。

```
#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 主函数main (见Smpl_DrvPDMA.c)

(如果仅要关注PDMA传输过程，请跳过本章节)

主函数先通过 “SystemFrequecy”设定系统时钟频率，然后用函数“DrvUART_Open()”打开两个串口，最后通过调用PDMA_UART()函数启动整个PDMA流程。

在下面的代码里，打开第一个UART 端口 (UART_PORT0)，将用来在控制台上输入输出信息；打开第二个UART端口(UART_PORT1) ，将被PDMA用来和内存之间传输数据。

打开两个UART的代码:

```
/* 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 PDMA 传输流程(见 Smpl_DrvPDMA.c)

2.3.1 进程中的流程

函数PDMA_UART() 见证了整个PDMA的传输过程。

开始传输之前，需要先做一些数据准备和初始化的动作：

1. 调用BuildSrcPattern() 将要传输的数据填写到SrcArray中;
2. 调用DrvPDMA_Init() 初始化PDMA 模块;
3. 用参数eDRVPDMA_UART1和eDRVPDMA_UART0调用DrvPDMA_SetCHForAPBDevice()，这样UART1/UART0就和PDMA设备联系到了一起。

```
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);
```

在上面的初始化做完以后，分别设定两路通道，用于PDMA写设备和读设备。这一步同时也设定了传输的地址、数据宽度等信息。

(请留意UART1 传输和发送寄存器事实上是同一地址，这里都写作UARTPort。)

```
/* 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);
```

接下来挂再PDMA中断函数

```
/* 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 );
```

到这里真正的传输就要开始了！对于UART设备，需要额外调用一个函数DrvUART_SetPDMA()，用来启动PDMA模式。对于内存到内存的PDMA, 不需要调用 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);
```

当PDMA 传输完成，系统将触发中断处理函数PDMA0_Callback()，这个函数会设定一个标记。 主流程 PDMA_UART() 检查到这个标记的状态改变就会返回。

2.3.2 中断处理函数中的流程

当PDMA的通道0一次传输结束，就会触发中断处理函数PDMA0_Callback()。如下面的代码所示，中断处理函数再次启动了新一轮的PDMA 传输。当总共传输次数到达10 次时，标记IsTestOver 会被射成TRUE，告诉进程所有传输都已经结束了。

```
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 调用过程

3.1 进程中的运行步骤

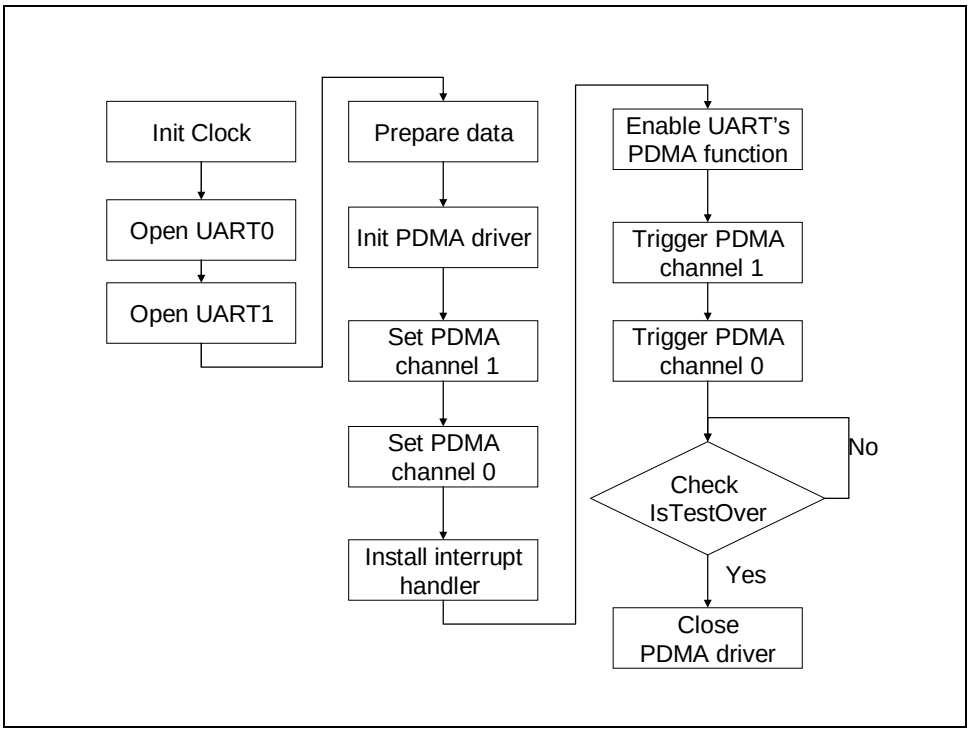

Figure 3-1 进程中的运行步骤

1. 初始化始终设定
2. 打开两个UART 设备：UART0 和 UART1
3. 准备要传输的数据
4. 初始化 PDMA 驱动
5. 将 PDMA 通道1设定成内存到UART1

6. 将 PDMA 通道0设定成UART1到内存
7. 挂载PDMA通道0的中断处理函数
8. 设定UART支持 PDMA 方式
9. 触发PDMA 的通道1，开始向UART1写数据.
10. 触发PDMA 的通道0，开始从UART1读数据.
11. 等待中断设定IsTestOver标记.
12. 结束PDMA驱动

3.2 中断中的运行步骤

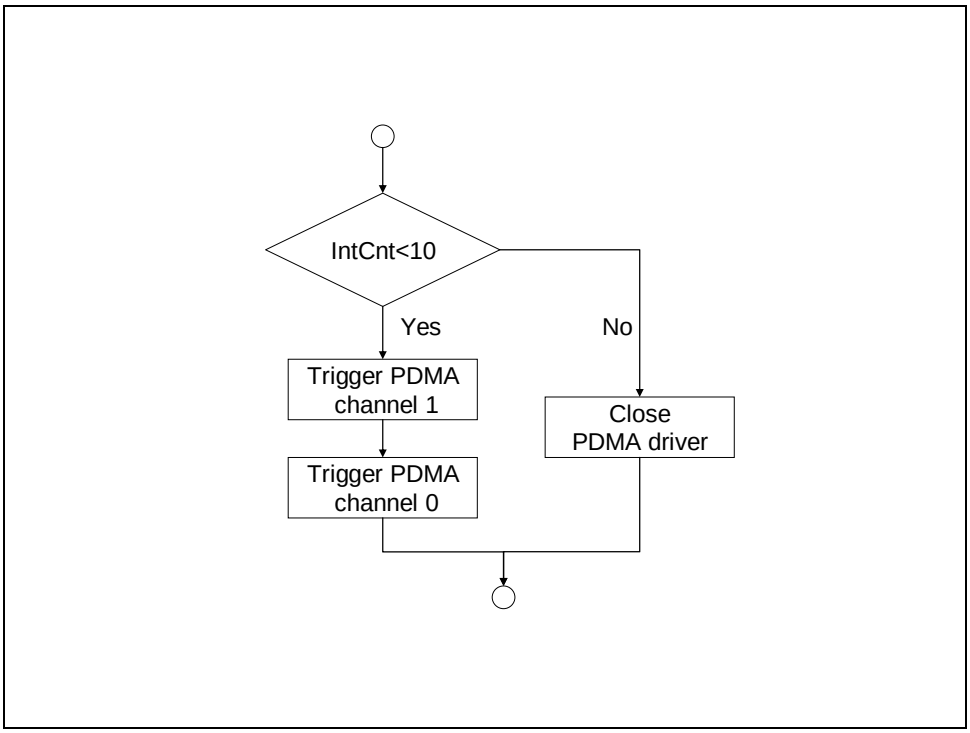

Figure 3-2 终端中的运行步骤

1. 如果传输次数少于10次
   a) 触发PDMA 的通道1，开始向UART1写数据.
   b) 触发PDMA 的通道0，开始从UART1读数据.
   c) 传输次数变量 (IntCnt) 加 1.
2. 否则(传输次数达到10次)
   a) 将标记IsTestOver 设定为TRUE.

3.3 API 用法参考

- 请参考NUC1xx Driver Reference Guide.doc

4 运行环境和结果

4.1 测试 Smpl_DrvPDMA

您可以用Keil MDK编译并下载运行该PDMA 示例代码(Smpl_DrvPDMA)。在Keil MDK下运行的时候，需要一块NUC1xx 系列开发板和一个ICE。.如果程序已经烧录在Flash中了，用户重启动开发板后，也能直接看到示例程序运行的结果。

4.2 结果

程序运行的时候，SrcArray中的源数据将被写到UART1 Transmit Holding寄存器中；UART1 Receive Buffer寄存器中的数据，将被写到目标数据DestArray中。通过这种方式，SrcArray中的数据会被复制到DestArray中。用户通过检查DestArray中的内存，就可以验证PDMA的传输结果。

版本历史

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.