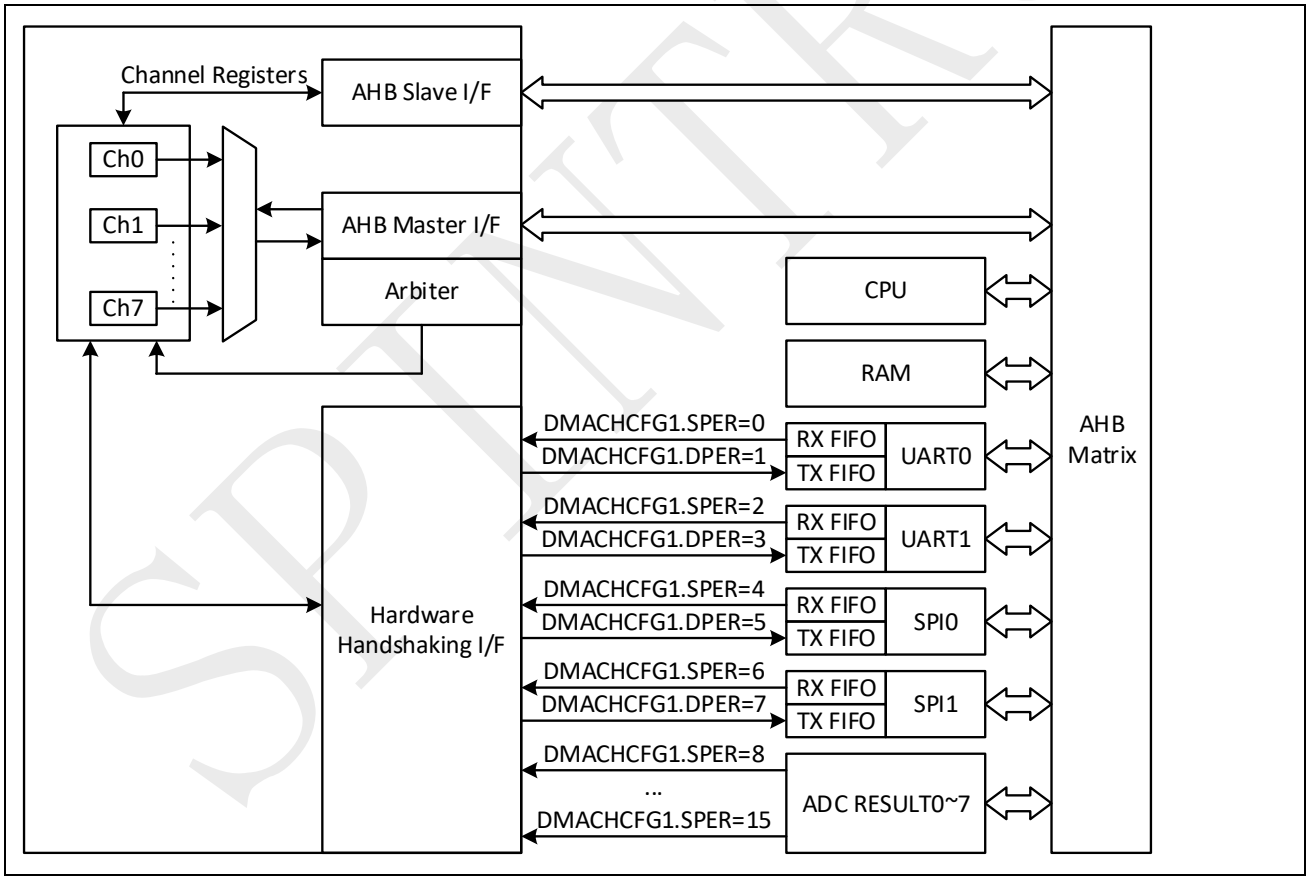
SPC1169 DMA 使用指南

版本 A/0 – 2023 年 5 月

概述

SPD1179/SPD1176 中的 DMA 控制器总共有 8 个通道，用于管理来自一个或多个外设的内存访问请求，其内部仲裁器用于处理 DMA 请求之间的优先级。DMA 控制器通过总线矩阵与 RAM、ADC、UART 和 SPI 相互连接，框图如图 1-1：DMA 框图所示。

图 1-1：DMA 框图



目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 3 月 24 日	CanChai	Released	首次发布。

术语或缩写

术语或缩写	描述
AHB	高级高性能总线
Burst Transfer	连续传输

1 DMA 用于内存之间的数据传输

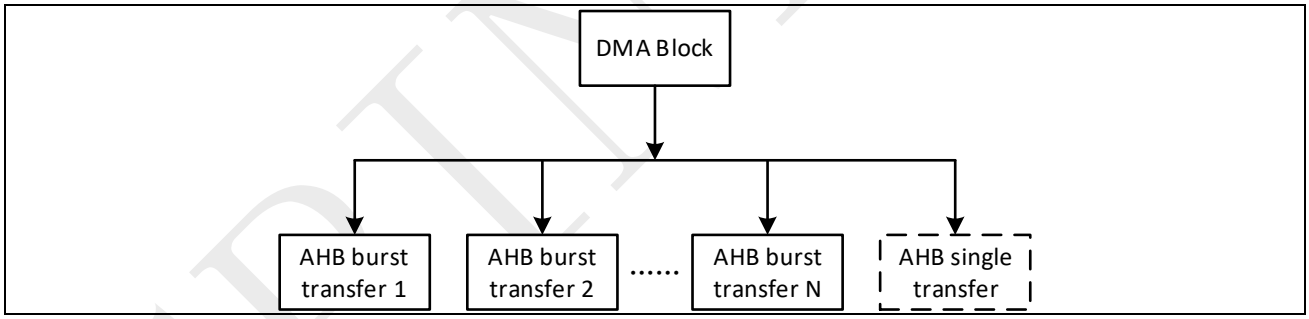
1.1 传输逻辑理解

在与内存之间传输数据时，DMA 仅支持单数据块传输，单数据块的数量可以通过 DMACHCTL1.BLKTS 寄存器进行配置。如图 1-1：DMA 框图所示，DMA 即可作为主设备与 AHB 总线沟通，也可作为从设备与 AHB 总线沟通，从 AHB 总线的角度来看，对于 DMA 和内存之间的传输，其需要传输的块数据在 AHB 上将会被被直接分解成 AHB Burst（连续）传输以及 AHB 单次传输的序列，如图 1-1：DMA 数据块传输在 AHB 总线上的拆分所示。

一个 AHB Burst 中包含了多个 AHB 单次传输，AHB 单次传输的长度通过寄存器设置（在 SDK 中可通过分别调用 DMA_SetSourceTransferWidth 及 DMA_SetDestinationTransferWidth 接口对源端及目的端进行单次传输长度的设置），需要特别注意的是，源端和目的端的单次传输长度必须设置一样，否则传输将会出错。AHB Burst 每次传输的大小也可通过设置 DMA 的寄存器进行设置（在 SDK 中可通过分别调用 DMA_SetSourceBurstLen 及 DMA_SetDestinationBurstLen 接口对源端及目的端进行 Burst 传输长度的设置）。

图 1-1：DMA 数据块传输在 AHB 总线上的拆分中（SDK 中已提供相应的接口），当设置的数据块大小（DMA Block）不是 AHB Burst 传输长度的倍数时，如图 1-1：DMA 数据块传输在 AHB 总线上的拆分所示，则需要一系列 AHB Burst 传输后跟 AHB 单次传输才能完成整个数据块传输。

图 1-1：DMA 数据块传输在 AHB 总线上的拆分



1.2 示例代码

以下代码作用是在两块内存之间搬移数据：源端有 3 个 32bit 的数据，需要搬移到目的地址，因此在代码中进行了如下设置：

- 设置源端和目的端地址；
- 设置传输类型为内存到内存；
- 设置源端和目的端地址增长模式为向上增长（也即每进行完一次单次传输后，将源端和目的端地址增加单次传输位宽的字节数，例如，假如单次传输的位宽为字，则每次传输完一个单次传输，源端和目的端地址+4）；
- 设置源端和目的端单次传输位宽；
- 设置此次 DMA 数据块传输大小为 3；
- 使能 DMA 中断；
- 使能 DMA，开始数据传输；
- DMA 传输完成之后，将会进入中断，并打印目的端收到的数据。

Example Code

```
#include <stdio.h>
#include "spd1179.h"

#define                 SOURCE_ADDRESS              0x1ffffa00
#define                 TARGET_ADDRESS              0x1ffffc00


uint32_t                *pu32Src                    = (uint32_t *)SOURCE_ADDRESS;
uint32_t                *pu32Dst                    = (uint32_t *)TARGET_ADDRESS;
uint32_t                i;      /* Print Num */

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Enable global INT for DMA */
    NVIC_EnableIRQ(DMAC_IRQn);

    /* Init source memory */
    pu32Src[0] = 0x22334411;
    pu32Src[1] = 0x55334411;
    pu32Src[2] = 0x88334411;

    /* Enable DMA */
```

```
    DMA_Enable();

    /* Source address */
    DMA_SetSourceAddr(DMACH0, SOURCE_ADDRESS);

    /* Destination address */
    DMA_SetDestinationAddr(DMACH0, TARGET_ADDRESS);

    /* Transfer type */
    DMA_SetTransferType(DMACH0, DMA_MEMORY_TO_MEMORY);

    /* Source address increment */
    DMA_SetSourceAddrMode(DMACH0, DMA_ADDRESS_MODE_INCREASE);

    /* Destination address increment */
    DMA_SetDestinationAddrMode(DMACH0, DMA_ADDRESS_MODE_INCREASE);

    /* Set Source Transfer Width */
    DMA_SetSourceTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);

    /* Set Destination Transfer Width */
    DMA_SetDestinationTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);

    /* Block transfer num */
    DMA_SetBlockTransferSize(DMACH0, 3);

    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(DMACH0);

    /* Enable Channel 0 transfer complete interrupt */
    DMA_EnableTransferCompleteInt(DMA_CH0);

    /* Enable Channel 0 */
    DMA_EnableChannelTransfer(DMA_CH0);

    while (1)
    {

    }
}


void DMAC_IRQHandler(void)
{
    /* Get TRANSFER ERROR flag */
    if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_ERROR) != 0)
    {
        printf("DMA Channel Error!\n");
    }

    /* Get TRANSFER COMPLETE flag */
    if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_COMPLETE) != 0)
    {
        printf("Channel transfer complete!\n");
    }

    /* Get DMA_CH0 TRANSFER COMPLETE flag */
    if (DMA_GetTransferCompleteIntFlag(DMA_CH0) != 0)
```

```
    {
        printf("Channel 0 transfer complete!\n");

        for (i = 0; i < 3; i++)
        {
            printf("Dest[%d] = 0x%08X\n", i, pu32Dst[i]);
        }

        /* Clear DMA_CH0 TRANSFER COMPLETE flag */
        DMA_ClearTransferCompleteInt(DMA_CH0);
    }
}
```

1.3 特别说明

本小节将对一些常见的疑惑加以解释：

假如源端有 3 个 8bit 的数需要搬移到目的地址，但是此时误将单次传输位宽设置为 WORD，且将数据块大小设置为 3，当 DMA 开始搬移数据时，仍然会按照每次一个 WORD 的位宽搬移三次数据，也即共搬移了 4*3=8 bytes 的数据，源端的 3 个 8bit 的数据包含在里面，其它数据将是其它未知数据。

会发生此现象的原因是，DMA 是硬件，在内存之间搬移数据时，它并不去检查地址越界等问题，而是按照设定好的数值机械搬移数据，所以在这种情况下，需要格外注意单次传输位宽，数据块大小等数据的设置。

2 DMA 用于内存与外设之间的数据传输

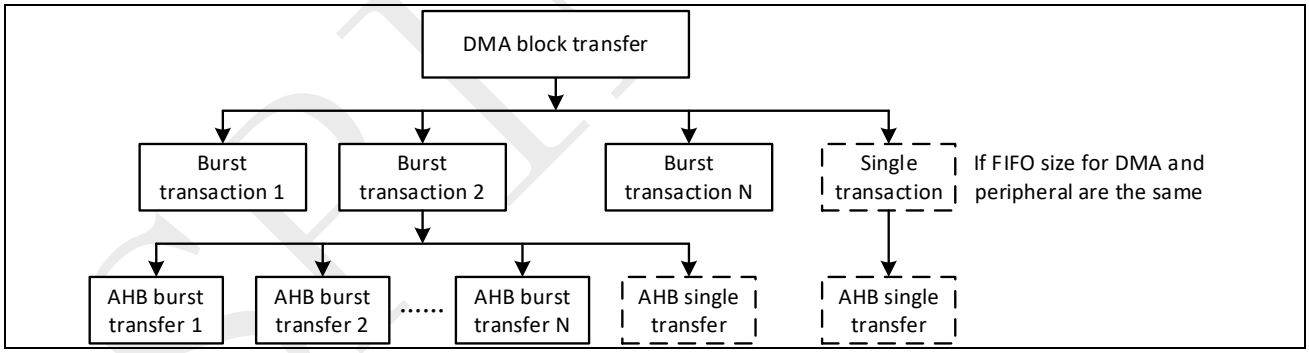
2.1 传输逻辑理解

如图 2-1：DMA 数据块传输在 AHB 总线上的拆分所示，在内存与外设之间传输数据时，数据块首先被分成若干个单个或连续的传输事务，并由外设发起。然后，每个传输事务再被转换为 AHB 传输（单个或连续），以与非内存型外设进行通信。图中数据块的数量可以通过 DMACHCTL1.BLKTS 寄存器进行配置。如图 1-1：DMA 框图所示，DMA 即可作为主设备与 AHB 总线沟通，也可作为从设备与 AHB 总线沟通，从 AHB 总线的角度来看，单个或连续的传输事务会进一步分解成 AHB Burst 传输以及 AHB 单次传输的序列，正如图 2-1：DMA 数据块传输在 AHB 总线上的拆分所示。并不用太过关心数据块是怎么拆分成单个或者连续传输事务，只需了解数据块将会分解成传输事务，传输事务会进一步拆分成 AHB 总线上的传输即可，需要关心的是，AHB 上的单次传输及连续传输与寄存器设置之间的关系。

一个 AHB Burst 中包含了多个 AHB 单次传输，AHB 单次传输的长度通过寄存器设置（在 SDK 中可通过分别调用 DMA_SetSourceTransferWidth 及 DMA_SetDestinationTransferWidth 接口对源端及目的端进行单次传输长度的设置），需要特别注意的是，源端和目的端的单次传输长度必须设置一样，否则传输将会出错。AHB Burst 每次传输的大小也可通过设置 DMA 的寄存器进行设置（在 SDK 中可通过分别调用 DMA_SetSourceBurstLen 及 DMA_SetDestinationBurstLen 接口对源端及目的端进行 Burst 传输长度的设置）。

图 2-1：DMA 数据块传输在 AHB 总线上的拆分中（SDK 中已提供相应的接口），当连续传输事务大小不是 AHB Burst 传输长度的倍数时，则需要一系列 AHB Burst 传输后跟 AHB 单次传输才能完成整个数据块传输。

图 2-1：DMA 数据块传输在 AHB 总线上的拆分



2.2 示例代码

以下代码作用是将内存中的数据，通过 UART TX 发送，并通过 UART 的 LoopBack 功能回到 UART RX，在 DMA 传输数据结束后，在 DMA 中断中打印 UART RX 接收到的数据，因此在代码中进行了如下设置：

- UART 相关设置：

  a.初始化 UART（设置时钟，初始化波特率，设置 I/O 等）；

  b.设置 UART 数据位宽为 32bit；

  c.使能 UART LoopBack 功能；

  d.使能 UART TX DMA 功能，并关闭 UART RX DMA 功能；

- 使能 DMA，使能中断；
- 设置传输类型为内存至非内存型外设；
- 设置源端及目的端单次传输位宽为 WORD，且连续传输时的传输长度为 1 个 WORD；
- 设置数据块大小为 4；
- 设置源端为内存地址，目的端为 UART TX；
- 设置源端地址增长模式为向上增长（也即每进行完一次单次传输后，将源端和目的端地址增加单次传输位宽的字节数，例如，假如单次传输的位宽为字，则每次传输完一个单次传输，源端和目的端地址+4）；
- 设置目的端地址增长模式为不改变，也即保持目的端地址不变；
- 设置源端和目的端的握手类型为硬件类型，且极性为高；
- 设置 DMA 目的端非内存型外设为 UART 的 TX；
- 使能 DMA 通道传输功能，开始数据传输；

  DMA 传输完成之后，将会进入中断，并打印目的端收到的数据。

Example Code

```
#include <stdio.h>
#include "spd1179.h"

#define                 SOURCE_ADDRESS              0x1ffffa00


uint32_t                *pu32Src                    = (uint32_t *)SOURCE_ADDRESS;
uint32_t                au32Result[4];                                          /* Result data */
uint32_t                i;                                                       /* Print Num */


void DMA_UART_Init(UART_REGS *UARTx)
{
    /* Unlock Key Protect */
    UART_WALLOW(UARTx);

    /* Stop UART */
    UART_Disable(UARTx);
    UART_DisableSIR(UARTx);
```

```
    LIN_Disable(UARTx);

    /* UART_Init */
    UART_Init(UARTx, 38400);
    UART_ClearTxFIFO(UARTx);
    UART_ClearRxFIFO(UARTx);

    /* Set UART BusWidth */
    UART_Set32BitBusWidth(UARTx);

    /* Enable UART loop */
    UART_EnableLoopback(UARTx);

    /* Enable the Tx DMA */
    UART_SetTxDMA(UARTx, ENABLE);
    UART_SetRxDMA(UARTx, DISABLE);
}


void DMA_Enable_Int_Init(DMA_ChannelEnum DMA_CHx)
{
    /* Enable DMA */
    DMA_Enable();

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMA_CHx);
    DMA_EnableTransferErrorInt(DMA_CHx);

    /* Clear all interrupt flag */
    DMA_ClearTransferCompleteInt(DMA_CHx);
    DMA_ClearTransferErrorInt(DMA_CHx);

    /* Disable the DMA transfer */
    DMA_DisableChannelTransfer(DMA_CHx);
}


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Enable global INT for DMA */
    NVIC_EnableIRQ(DMAC_IRQn);

    /* Stop DMA */
    DMA_Disable();
```

```
    /* Wait until DMA is disable */
    while (DMA_IsEnable() == 1)
    {}

    /* DMA relevant UART parameter set */
    DMA_UART_Init(UART1);

    /* Enable UART */
    UART_Enable(UART1);

    /* DMA enable interrupt */
    DMA_Enable_Int_Init(DMA_CH0);

    /* Transfer type */
    DMA_SetTransferType(DMACH0, DMA_MEMORY_TO_PERIPHERAL);

    /* Set source transfer parameters */
    DMA_SetSourceTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);
    DMA_SetSourceBurstLen(DMACH0, DMA_BURST_LENGTH_1_WORD);

    /* Set destination transfer parameters */
    DMA_SetDestinationTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);
    DMA_SetDestinationBurstLen(DMACH0, DMA_BURST_LENGTH_1_WORD);

    /* Block transfer num */
    DMA_SetBlockTransferSize(DMACH0, 4);

    /* Init source memory */
    pu32Src = (uint32_t *)SOURCE_ADDRESS;
    pu32Src[0] = 0x82131415;
    pu32Src[1] = 0x86171819;
    pu32Src[2] = 0x80212223;
    pu32Src[3] = 0x84252627;

    for (i = 0; i < 4; i++)
    {
        printf("The Send word is %x\n", *((uint32_t *)pu32Src + i));
    }

    /* Source address */
    DMA_SetSourceAddr(DMACH0, SOURCE_ADDRESS);
    /* Source address increment */
    DMA_SetSourceAddrMode(DMACH0, DMA_ADDRESS_MODE_INCREASE);
    DMA_SetSourceHandShake(DMACH0, DMA_HANDSHAKE_BY_SOFTWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH) ;

    /* Destination address */
    DMA_SetDestinationAddr(DMACH0, (uint32_t)(&UART1->UARTDAT));
    /* Destination address increment */
    DMA_SetDestinationAddrMode(DMACH0, DMA_ADDRESS_NO_CHANGE);
    DMA_SetDestinationHandShake(DMACH0, DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH) ;
    /* Destination peripheral */
    DMA_SetDestinationPeripheral(DMACH0, DMA_DPER_UART1_TX);

    /* DMA disable suspend */
    DMA_DisableSuspend(DMACH0);
```

```
    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(DMACH0);

    while (1)
    {
        /* Enable Channel 0 transfer*/
        DMA_EnableChannelTransfer(DMA_CH0);

        /* Wait until all data is received */
        while ((UART_GetRxFIFOLevel(UART1) != 16))
        {
        }

        for (i = 0; i < 4; i = i + 1)
        {
            au32Result[i] = UART_ReadWord(UART1);
            printf("%d:Rx 0x%08X\n", i, au32Result[i]);
        }

        Delay_Ms(100);

        /* Reset the source addr, because source addr is increased after previous loop */
        DMA_SetSourceAddr(DMACH0, SOURCE_ADDRESS);
    }
}


void DMAC_IRQHandler(void)
{
    /* Get TRANSFER ERROR flag */
    if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_ERROR) != 0)
    {
        printf("DMA Channel Error!\n");
    }

    /* Get TRANSFER COMPLETE flag */
    if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_COMPLETE) != 0)
    {
        printf("Channel transfer complete!\n");
    }

    /* Get DMA_CH0 TRANSFER COMPLETE flag */
    if (DMA_GetTransferCompleteIntFlag(DMA_CH0) != 0)
    {
        printf("Channel 0 transfer complete!\n");
        /* Clear DMA_CH0 TRANSFER COMPLETE flag */
        DMA_ClearTransferCompleteInt(DMA_CH0);
    }
}
```

2.3 特别说明

本小节将对一些常见的疑惑加以解释：

在本章节的示例代码中，设置 DMA 的源端是内存中的地址，而目的端是 UART 的 DATA 寄存器，且使能的是 TX，关闭了 RX，其原因是，示例想要实现的功能是把内存中的数据通过 UART TX 发送出去，所以源端是内存地址，而目的端是 UART DATA 寄存器，且 DATA 中的数据时来自 UART TX。

至于为什么有 UART RX/TX 的 DMA 使能和关闭接口，其目的是为了能让用户主动控住这些外设与 DMA 的交互功能。

3 DMA 用于外设与外设之间的数据传输

3.1 传输逻辑理解

此种类型的传输，在 AHB 级的传输理解同 2.1 小节，在此不再赘述。

3.2 示例代码

以下代码作用是将数据通过 SPI 发送到 UART，代码设置如下：

- SPI 相关设置：
  a.初始化 SPI（设置 SPI 工作模式，频率等）；
  b.使能 SPI LoopBack；
  c.设置 RX FIFO 阈值为 3；
  d.使能 SPI RX DMA 功能，关闭 TX DMA 功能；
- UART 相关设置：
  a.初始化 UART（设置时钟，初始化波特率，设置 I/O 等）；
  b.设置 UART 数据位宽为 32bit；
  c.使能 UART LoopBack 功能；
  d.使能 UART TX DMA 功能，并关闭 UART RX DMA 功能；
- 使能 DMA，使能中断；
- 设置传输类型为非内存型外设至非内存型外设；
- 设置源端及目的端单次传输位宽为 WORD，且连续传输时的传输长度为 1 个 WORD；
- 设置源端为 SPI DATA 寄存器，目的端为 UART DATA 寄存器；
- 设置数据块大小为 4；
- 设置源端和目的端地址增长模式为不改变，也即保持地址不变；
- 设置源端和目的端的握手类型为硬件类型，且极性为高；
- 设置 DMA 目的端非内存型外设为 UART 的 TX；
- 使能 DMA 通道传输功能，开始数据传输；

Example Code

```
#include <stdio.h>
#include "spd1179.h"


uint32_t                    au32Src[4];                 /*
Source data */
uint32_t                    au32Result[4];              /*
Result data */
uint32_t                    i;                           /*
Print Num */
```

```
void DMA_SPI_Init(SPI_REGS *SPIx)
{
    /* Unlock Key Protect */
    SPI_WALLOW(SPIx);

    /* Stop SPI */
    SPI_Disable(SPIx);

    /* SPI_Init */
    SPI_Init(SPIx, SPI_MASTER, SPI_MODE_PHS_POL_00, 32, 1000000);

    /* Enable SPI loop */
    SPI_EnableLoopback(SPIx);

    /* Set the Rx FIFO Threshold, the DMA will start next transfer when the Rx current deep > Rx FIFO
Threshold*/
    SPI_SetRxFIFOThreshold(SPIx, 3);

    /* Enable the Rx DMA */
    SPI_SetTxDMA(SPIx, DISABLE);
    SPI_SetRxDMA(SPIx, ENABLE);
}


void DMA_UART_Init(UART_REGS *UARTx)
{
    /* Unlock Key Protect */
    UART_WALLOW(UARTx);

    /* Stop UART */
    UART_Disable(UARTx);
    UART_DisableSIR(UARTx);
    LIN_Disable(UARTx);

    /* UART_Init */
    UART_Init(UARTx, 38400);
    UART_ClearTxFIFO(UARTx);
    UART_ClearRxFIFO(UARTx);

    /* Set UART BusWidth */
    UART_Set32BitBusWidth(UARTx);

    /* Enable UART loop */
    UART_EnableLoopback(UARTx);

    /* Enable the Tx DMA */
    UART_SetTxDMA(UARTx, ENABLE);
    UART_SetRxDMA(UARTx, DISABLE);
}


void DMA_Enable_Int_Init(DMA_ChannelEnum DMA_CHx)
{
    /* Enable DMA */
    DMA_Enable();

    /* Enable the DMA interrupt */
```

```
    DMA_EnableTransferCompleteInt(DMA_CHx);
    DMA_EnableTransferErrorInt(DMA_CHx);

    /* Clear all interrupt flag */
    DMA_ClearTransferCompleteInt(DMA_CHx);
    DMA_ClearTransferErrorInt(DMA_CHx);

    /* Disable the DMA transfer */
    DMA_DisableChannelTransfer(DMA_CHx);
}

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Enable global INT for DMA */
    NVIC_EnableIRQ(DMAC_IRQn);

    /* Stop DMA */
    DMA_Disable();

    /* Wait until DMA is disable */
    while (DMA_IsEnable() == 1)
    {}

    /* DMA relevant SPI parameter set */
    DMA_SPI_Init(SPI0);

    /* Run SPI */
    SPI_Enable(SPI0);

    /* DMA relevant UART parameter set */
    DMA_UART_Init(UART1);

    /* Enable UART */
    UART_Enable(UART1);

    /* DMA enable interrupt */
    DMA_Enable_Int_Init(DMA_CH0);

    /* Transfer type */
    DMA_SetTransferType(DMACH0, DMA_PERIPHERAL_TO_PERIPHERAL);

    /* Set source transfer parameters */
    DMA_SetSourceTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);
    DMA_SetSourceBurstLen(DMACH0, DMA_BURST_LENGTH_4_WORDS);
```

```
    /* Set destination transfer parameters */
    DMA_SetDestinationTransferWidth(DMACH0, DMA_TRANSFER_IN_WORD);
    DMA_SetDestinationBurstLen(DMACH0, DMA_BURST_LENGTH_4_WORDS);

    /* Block transfer num */
    DMA_SetBlockTransferSize(DMACH0, 4);

    /* Init source memory */
    au32Src[0] = 0x82131415;
    au32Src[1] = 0x86171819;
    au32Src[2] = 0x80212223;
    au32Src[3] = 0x84252627;

    for (i = 0; i < 4; i++)
    {
        printf("The Send word is %x\n", au32Src[i]);
    }

    /* Source address */
    DMA_SetSourceAddr(DMACH0, (uint32_t)(&SPI0->SPIDATA));
    /* Source address increment */
    DMA_SetSourceAddrMode(DMACH0, DMA_ADDRESS_NO_CHANGE);
    DMA_SetSourceHandShake(DMACH0, DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH);
    /* Source peripheral */
    DMA_SetSourcePeripheral(DMACH0, DMA_SPER_SPI0_RX) ;

    /* Destination address */
    DMA_SetDestinationAddr(DMACH0, (uint32_t)(&UART1->UARTDAT));
    /* Destination address increment */
    DMA_SetDestinationAddrMode(DMACH0, DMA_ADDRESS_NO_CHANGE);
    DMA_SetDestinationHandShake(DMACH0, DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH);
    /* Destination peripheral */
    DMA_SetDestinationPeripheral(DMACH0, DMA_DPER_UART1_TX);

    /* DMA disable suspend */
    DMA_DisableSuspend(DMACH0);

    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(DMACH0);

    while (1)
    {
        /* Enable Channel 0 transfer*/
        DMA_EnableChannelTransfer(DMA_CH0);

        for (i = 0; i < 4; i = i + 1)
        {
            SPI_SendData(SPI0, au32Src[i]);
        }

        /* Wait until all data is received */
        while ((UART_GetRxFIFOLevel(UART1) != 16))
        {
        }

        for (i = 0; i < 4; i = i + 1)
```

```
            {
                au32Result[i] = UART_ReadWord(UART1);
                printf("%d:Rx 0x%08X\n", i, au32Result[i]);
            }

            Delay_Ms(100);
        }
    }

    void DMAC_IRQHandler(void)
    {
        /* Get TRANSFER ERROR flag */
        if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_ERROR) != 0)
        {
            printf("DMA Channel Error!\n");
        }

        /* Get TRANSFER COMPLETE flag */
        if (DMA_GetGlobalIntFlag(DMA_INT_TRANSFER_COMPLETE) != 0)
        {
            printf("Channel transfer complete!\n");
        }

        /* Get DMA_CH0 TRANSFER COMPLETE flag */
        if (DMA_GetTransferCompleteIntFlag(DMA_CH0) != 0)
        {
            printf("Channel 0 transfer complete!\n");

            /* Clear DMA_CH0 TRANSFER COMPLETE flag */
            DMA_ClearTransferCompleteInt(DMA_CH0);
        }
    }
```

3.3 特别说明

本小节将对一些常见的疑惑加以解释：

SDK 中为什么有 UART/SPI RX/TX 的 DMA 使能和关闭接口，其目的是为了能让用户主动控住这些外设与 DMA 的交互功能。