PDMA Driver Sample Code Reference Guide

V1.00.001

Publication Release Date: Sep. 2011

Support Chips:

ISD9160

Support Platforms:

NuvotonPlatform_Keil

Table of Contents

1 PDMA Driver Introduction 4
1.1 Feature 4
1.2 Limitation 4
2 Block Diagram 5
3 Calling Sequence 6
3.1 API Usage Reference 6
4 Code Section 7
4.1 Constant and Variable 7
4.2 Main Function 7
4.3 PDMA IRQ Handler 14
5 Execution Environment Setup and Result 16
6 Revision History 17

1 PDMA Driver Introduction

The ISD91XX incorporates a Peripheral Direct Memory Access (PDMA) controller that transfers data between SRAM and APB devices. The PDMA has four channels of DMA PDMA CH0~CH3). PDMA transfers are unidirectional and can be Peripheral-to-SRAM, SRAM-to-Peripheral or SRAM-to-SRAM. The peripherals available for PDMA transfer are SPI, UART, I2S, ADC and DPWM. PDMA operation is controlled for each channel by configuring a source and destination address and specifying a number of bytes to transfer. Source and destination addresses can be fixed, automatically increment or wrap around a circular buffer. When PDMA operation is complete, controller can be configured to provide CPU with an interrupt. This code “Smpl_DrvPDMA.c” shows the usage by PDMA of ISD9160.

1.1 Feature

- Demo memory to memory transfer by PDMA.
- Demo memory to APB transfer.
- Demo APB to memory transfer
- Interrupt method.

1.2 Limitation

- The PDMA has only 4 channels to operate.
- APB source can be SPI, UART, I2S, and ADC.
- APB destination can be SPI, UART, I2S, and DPWM.
- Max transfer width is 32 bits.
- Max transfer byte count is 0xFFFF (65535 in decimal).

2 Block Diagram

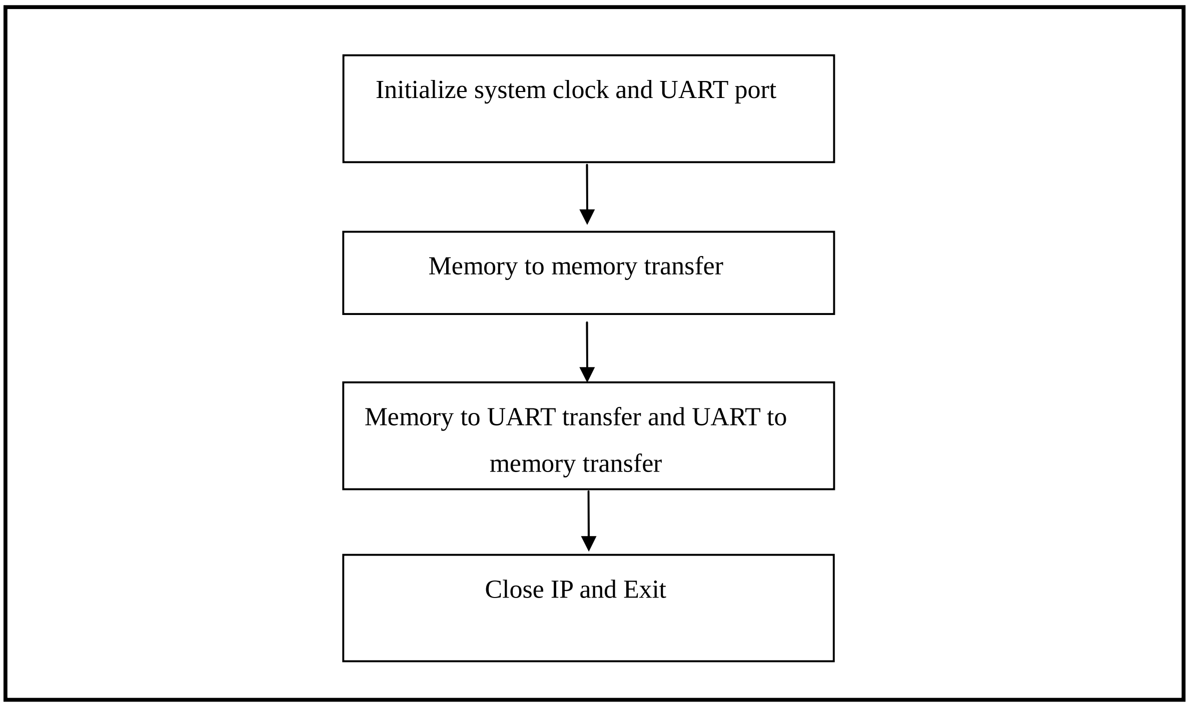
Initialize system clock and UART port
Memory to memory transfer
Memory to UART transfer and UART to memory transfer
Close IP and Exit

Figure 2-1: Main flow of sample code

3 Calling Sequence

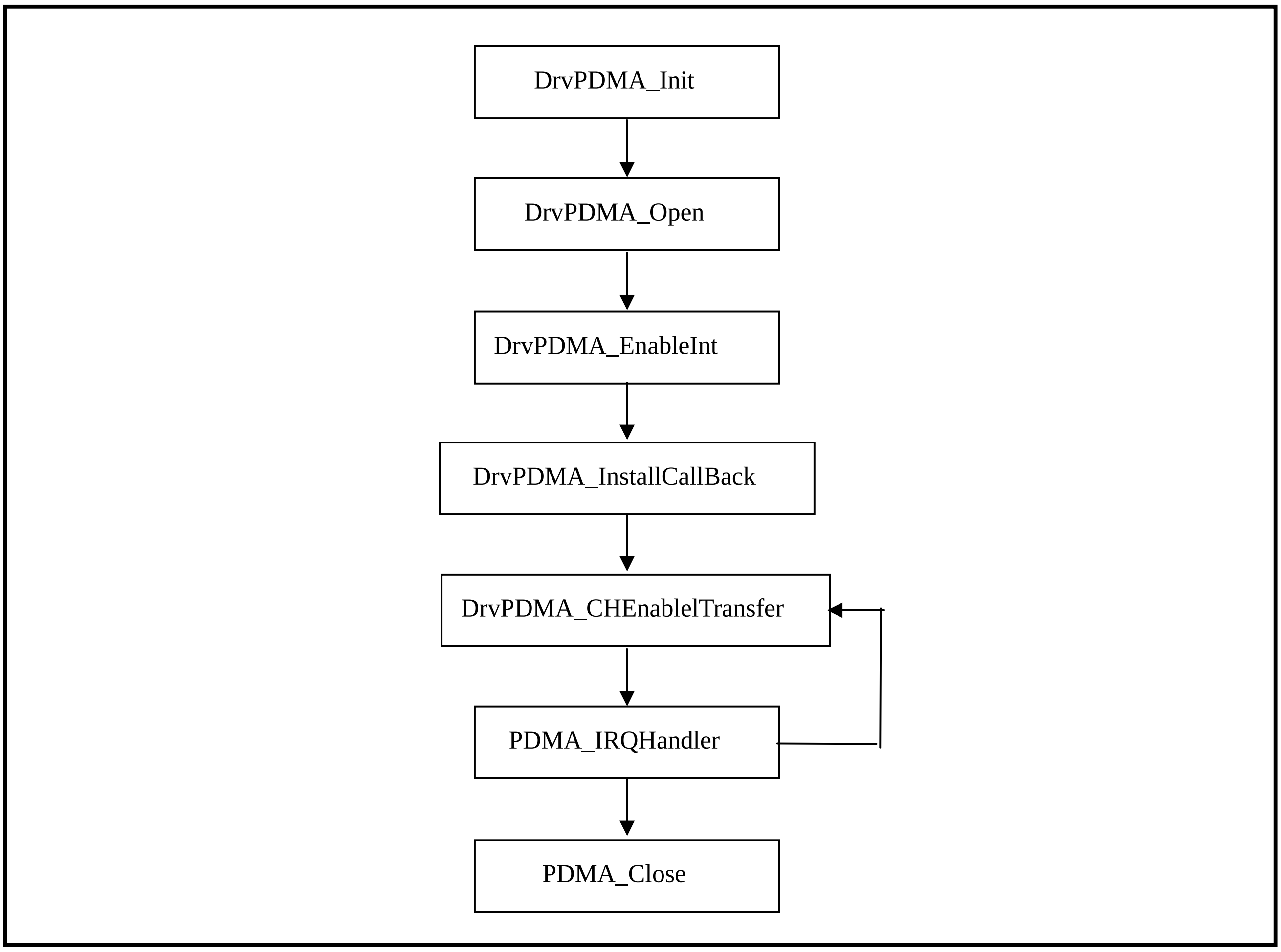

Figure 3-1: API call flow

3.1 API Usage Reference

- PDMA Driver User Guide.pdf

4 Code Section

4.1 Constant and Variable

UART_TEST_LENGTH defines the buffer length. SrcArray holds the data from PDMA to APB. DestArray holds the data received from APB to PDMA path. IntCnt sets up the test transfer cycles. IsTestOver is a software flag to tell the end of test transfer.

```
#define UART_TEST_LENGTH                    256
uint8_t SrcArray[UART_TEST_LENGTH];
uint8_t DestArray[UART_TEST_LENGTH];
int32_t IntCnt;
volatile int32_t IsTestOver;
```

4.2 Main Function

The InitialSystemClock() initialize system clock to 49.152MHz. The InitialUART() initialize UART port to 115200 baud, 8 bit, no parity, stop bit 1.

```
InitialSystemClock();
InitialUART();
```

The InitailSystemClock uses DrvSYS_SetHCLKSource(0) select internal oscillator for 49.152MHz.

```
void InitialSystemClock(void)
{
        /* Unlock the protected registers */
        UNLOCKREG();
        /* HCLK clock source */
        DrvSYS_SetHCLKSource(0);
        LOCKREG();
        /* HCLK clock frequency = HCLK clock source / (HCLK_N + 1) */
        DrvSYS_SetClockDivider(E_SYS_HCLK_DIV, 0);
}
```

InitialUART set up UART port configuration.

```
void InitialUART(void)
{
        STR_UART_T sParam;

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

        /* 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);
}
```

Show the description and call PDMA_MEM to make a transfer.

```
printf("+----------------------------------------------------------------------+\n");
printf("|                       PDMA Driver Sample Code                        |\n");
printf("|                                                                      |\n");
printf("+----------------------------------------------------------------------+\n");

printf("    This sample code will use PDMA to do memory to memory test. \n");
printf("    Test loopback 100 times \n");
printf("    press any key to continue ...\n");
getchar();

IsTestOver =FALSE;
PDMA_MEM();
printf("\n    PDMA MEM sample code is complete.\n\n");
```

Demo a memory to memory transfer. Build a source array and clear destination array. Set up PDMA source and destination address to the specified SrcArray and DestArray respectively. Use incremental mode to tell PDMA to increase address for every word transfer. Word width is 32bits. Open PDMA in channel 2.

```
void PDMA_MEM(void)
{
      STR_PDMA_T sPDMA;
     volatile uint32_t i;
      BuildSrcPattern((uint32_t)SrcArray,UART_TEST_LENGTH);

     ClearBuf((uint32_t)DestArray, UART_TEST_LENGTH,0xFF);

      /* PDMA Init */
     DrvPDMA_Init();

      /* PDMA Setting */

      /* CH1 TX Setting */
     sPDMA.u8TransWidth                = eDRVPDMA_WIDTH_32BITS;
      sPDMA.i32ByteCnt                 = UART_TEST_LENGTH;
      /* CH0 RX Setting */
      sPDMA.sSrcAddr.u32Addr           = (uint32_t)SrcArray;
     sPDMA.sDestAddr.u32Addr           = (uint32_t)DestArray;
      sPDMA.u8Mode                     = eDRVPDMA_MODE_MEM2MEM;
      sPDMA.sSrcAddr.eAddrDirection    = eDRVPDMA_DIRECTION_INCREMENTED;
      sPDMA.sDestAddr.eAddrDirection = eDRVPDMA_DIRECTION_INCREMENTED;
      DrvPDMA_Open(eDRVPDMA_CHANNEL_2,&sPDMA);
```

Specify the PDMA interrupt mode: issue when block transfer ends. Hook up PDMA2_Callback to PDMA IRQ_Handler. After all settings are ready, DrvPDMA_CHEnableTransfer() will start the PDMA block to move data from SrcArray to DestArray.

```
        /* Enable INT */
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_2, eDRVPDMA_BLKD );

        /* Install Callback function */
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_2, eDRVPDMA_BLKD,
        (PFN_DRVPDMA_CALLBACK) PDMA2_Callback );

        /* Trigger PDMA specified Channel */
        IntCnt = 0;
        IsTestOver=FALSE;
        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_2);

        /* Trigger PDMA 10 time and the S/W Flag will be change in PDMA callback funtion */
        while(IsTestOver==FALSE);

        /* Close PDMA Channel */
        DrvPDMA_Close();
}
```

Show the description and call PDMA_UART to make a transfer.

```
/* PDMA Sample Code: UART0 Tx/Rx Loopback */
printf("+------------------------------------------------------------------------+\n");
printf("|                       PDMA Driver Sample Code                          |\n");
printf("|                                                                        |\n");
printf("+------------------------------------------------------------------------+\n");
printf("    This sample code will use PDMA to do UART0 loopback test. \n");
printf("    Test loopback 10 times \n");
printf("    I/O configuration:\n");
printf("        GPA8 <--> GPA9\n\n");
printf("    press any key to continue ...\n");
getchar();

PDMA_UART();
printf("\n    PDMA UART sample code is complete.\n\n");
```

Demo a memory to APB and APB to memory transfer. Build a source array and clear destination array. Set up PDMA source and destination address to the specified SrcArray and DestArray respectively. Use incremental mode to tell PDMA to increase address for every word transfer. Word width is 8bits. Open PDMA in channel 1 for memory to UART and PDMA channel 0 for UART to memory..

```
void PDMA_UART(void)
{
        STR_PDMA_T sPDMA;
        uint32_t    UARTPort;
        BuildSrcPattern((uint32_t)SrcArray,UART_TEST_LENGTH);

        UARTPort = UART0_BA;
        ClearBuf((uint32_t)DestArray, UART_TEST_LENGTH,0xFF);

        /* PDMA Init */
       DrvPDMA_Init();

        /* PDMA Setting */
        DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_1,eDRVPDMA_UART0,eDRVPD
MA_WRITE_APB);
        DrvPDMA_SetCHForAPBDevice(eDRVPDMA_CHANNEL_0,eDRVPDMA_UART0,eDRVPD
MA_READ_APB);

        /* 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);
```

Specify the PDMA interrupt mode: issue when block transfer ends. Hook up PDMA0_Callback for APB to memory interrupt and PDMA1_Callback for memory to APB interrupt. After all settings are ready, DrvPDMA_CHEnableTransfer() will start the PDMA block to move data from SrcArray to APB by channel 1. When APB receives data, PDMA will move data from APB to DestArray by channel 0.

```
        /* 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);

        /* 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 );

        /* Enable UART PDMA and Trigger PDMA specified Channel */
        DrvUART_SetPDMA(UART_PORT0,ENABLE);

        IntCnt = 0;
        IsTestOver=FALSE;

        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_1);

        /* Trigger PDMA 10 time and the S/W Flag will be change in PDMA callback funtion */
        while(IsTestOver==FALSE);

        /* Close PDMA Channel */
        DrvPDMA_Close();
}
```

4.3 PDMA IRQ Handler

Use DrvPDMA_InstallCallBack to hook up PDMA2_Callback function for PDMA IRQ Handler.

```
DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_2, eDRVPDMA_BLKD,
        (PFN_DRVPDMA_CALLBACK) PDMA2_Callback );
```

Each time when PDMA interrupt occur, PDMA2_Callback will check if transfer end, otherwise enable PDMA transfer again.

```
void PDMA2_Callback()
{
        extern int32_t IntCnt;
        printf("\tMem Transfer Done %02d!\r",++IntCnt);

        if(IntCnt<100)
        {
                DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_2);
        }
        else
        {
                IsTestOver = TRUE;
        }
}
```

When memory to APB PDMA interrupt occurs, check for transfer cycle. If user still want to transfer call DrvPDMA_CHEnablelTransfer() again.

```
void PDMA0_Callback()
{
        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;
        }
}
```

5 Execution Environment Setup and Result

- Prepare ISD9160 EVB
- Connect UART to user’s host UART port by female to female cable.
- Compile it under Keil environment.
- Run the program and the result will show on host console window.

6 Revision History

Version	Date	Description
V1.00.01	Sep. 2011	Created

6. Revision History

Version	Date	Description
V1.00.01	Sep. 2011	Created