DPWM Driver Sample Code Reference Guide

V1.00.001

Publication Release Date: Sep. 2011

Support Chips:
ISD9160

Support Platforms:
NuvotonPlatform_Keil

The information in this document is subject to change without notice.

The Nuvoton Technology Corp. shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

This documentation may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from the Nuvoton Technology Corp.

Table of Contents

1. DPWM Driver Introduction

The ISD91XX series includes a differential Class D (PWM) speaker driver capable of delivering 1W into an 8Ω load at 5V supply voltage. The driver works by up-sampling and modulating a PCM input to differentially drive the SPK+ and SPK- pins. The speaker driver operates from its own independent supply VCCSPK and VSSSPK. This supply should be well decoupled as peak currents from speaker driver are large. The code Smpl_DrvDPWM shows the usage of DPWM of ISD9160.

1.1 Feature

- Using 16K sampling rate to demo.
- Demo PDMA for DPWM
- Demo different DPWM parameter
- Biquad filter for further signal processing

1.2 Limitation

- Cock source has 2 option:49.152MHz and 98.304MHz
- Must use PDMA for DPWM buffer transfer
- Support sampling rates are 8K, 16K 32K.
- Dither level: 0, 1, 2 bits.
- Dead time only on /off.
- Modulation Frequency: Max=1890462Hz Min=126031Hz

2. Block Diagram

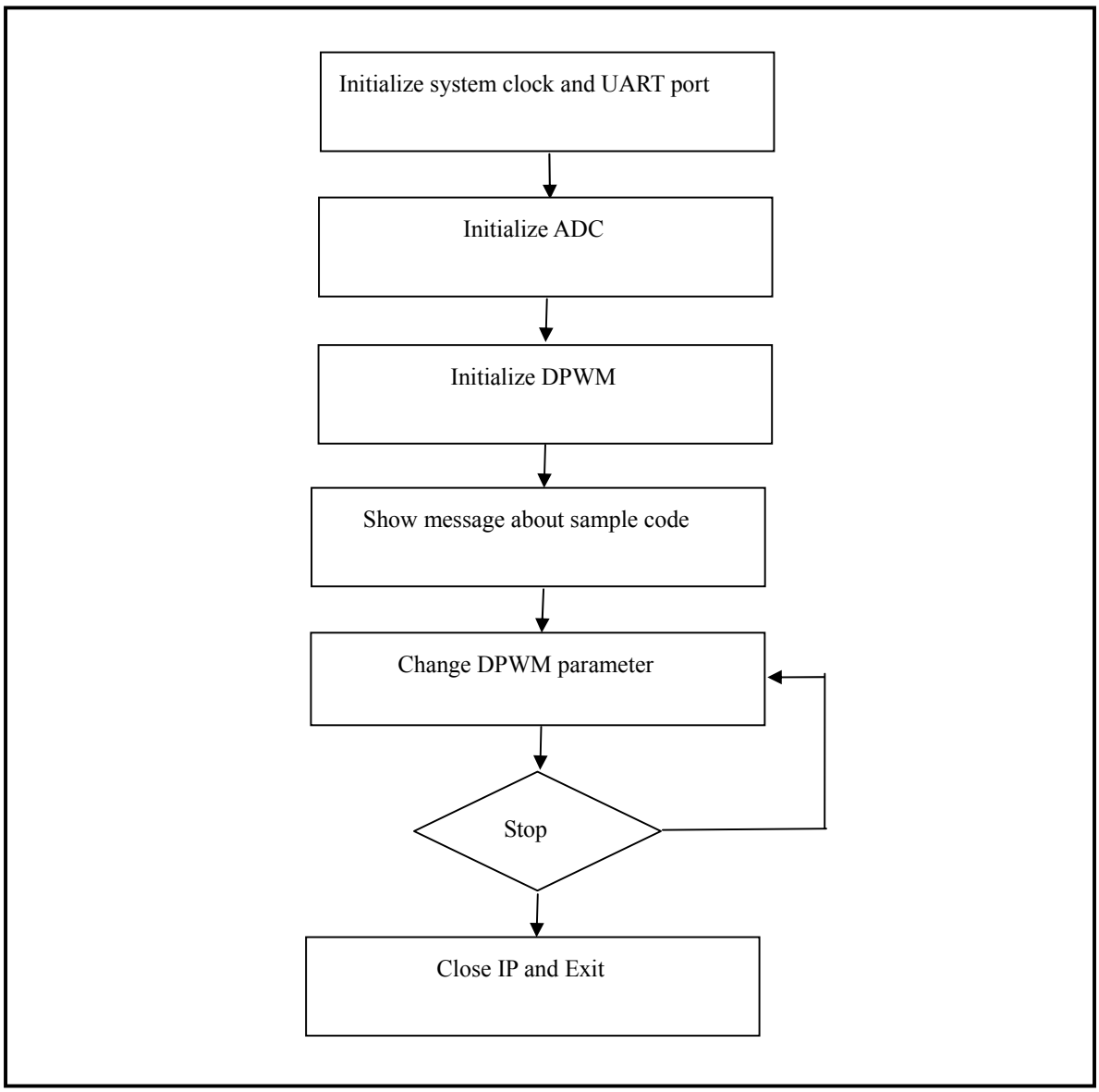

Figure 2-1: Main flow of sample code

3. Calling Sequence

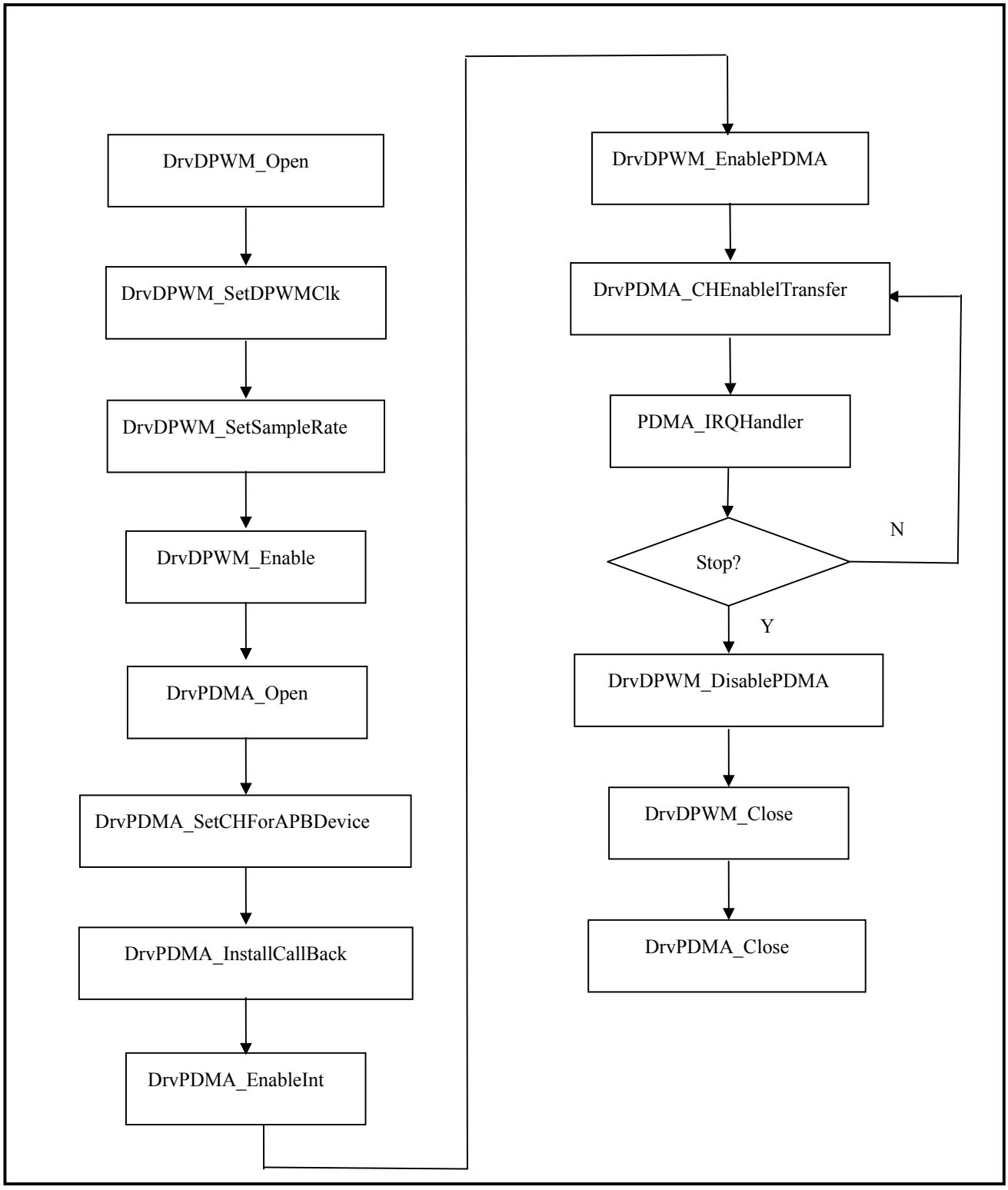

Figure 3-1: API call flow

3.1 API Usage Reference

- DPWM Driver User Guide.pdf
- PDMA Driver User Guide.pdf

4. Code Section

4.1 Constant and Variable

The MAX_FRAMESIZE defines one audio frame length by PDMA. The ZERO_BUF_SAMPLES defines a zero buffer length to transfer at initial phase of DPWM. DPWM_INITIALIZE is a status code to trigger DPWM callback initial behavior.

```
#define MAX_FRAMESIZE 160
__align(4) int16_t audio_buffer[4][MAX_FRAMESIZE];
#define ZERO_BUF_SAMPLES 4
__align(4) int16_t zero_buf[ZERO_BUF_SAMPLES]={0,0,0,0};

#define DPWM_INITIALIZE 0xaa5555aa
#define ADC_INITIALIZE 0xaa555555
#define I2S_INITIALIZE 0xaa55aa55
```

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. InitialACD() initialize microphone for input path and sampling rate is 16KHz. InitialDPWM() initialize DPWM for 16KHz operation.

```
InitialSystemClock();
InitialUART();
InitialADC();
InitialDPWM(16000);

printf("+------------------------------------------------------------------------+\n");
printf("|                    DPWM Driver Sample Code                             |\n");
printf("|                                                                        |\n");
printf("+------------------------------------------------------------------------+\n");
printf("    This sample code will use DPWM to play.\n");
printf("    DPWM Driver Version: %x\n", DrvDPWM_GetVersion());
printf("    Sample Rate set to %d Hz\n", DrvDPWM_GetSampleRate());
```

The InitialSystemClock 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);
}
```

InitialADC setup the analog path and select microphone as input. It also specifies the sampling rate by DrvADC_Open function.

```
S_DRVADC_PARAM sParam;

/* Open Analog block */
DrvADC_AnaOpen();

/* Power control */
DrvADC_SetPower(
                    eDRVADC_PU_MOD_ON,
                    eDRVADC_PU_IBGEN_ON,
                    eDRVADC_PU_BUFADC_ON,
                    eDRVADC_PU_BUFPGA_ON,
                    eDRVADC_PU_ZCD_ON);

/* PGA Setting */
DrvADC_PGAMute(eDRVADC_MUTE_PGA);
DrvADC_PGAUnMute(eDRVADC_MUTE_IPBOOST);
DrvADC_SetPGA(
                    eDRVADC_REF_SEL_VMID,
                    eDRVADC_PU_PGA_ON,
                    eDRVADC_PU_BOOST_ON,
                    eDRVADC_BOOSTGAIN_0DB);

DrvADC_SetPGAGaindB(0);    // 0 dB

/* MIC circuit configuration */
DrvADC_SetVMID(
                    eDRVADC_PULLDOWN_VMID_RELEASE,
                    eDRVADC_PDLORES_CONNECTED,
                    eDRVADC_PDHIRES_DISCONNECTED);
DrvADC_SetMIC(TRUE, eDRVADC_MIC_BIAS_90_VCCA);
```

```
/* Open ADC block */
sParam.u8AdcDivisor      = 0;
sParam.u8SDAdcDivisor = 16;
sParam.eOSR                  = eDRVADC_OSR_192;
sParam.eInputSrc      = eDRVADC_MIC;
sParam.eInputMode = eDRVADC_DIFFERENTIAL;
sParam.u8ADCFifoIntLevel = 7;
DrvADC_Open(&sParam);

DrvADC_PGAUnMute(eDRVADC_MUTE_PGA);
```

InitialDPWM open DPWM block and set the sampling rate.

```
void InitialDPWM(uint32_t SampleRate)
{
        DrvDPWM_Open();
        DrvDPWM_SetDPWMClk(E_DRVDPWM_DPWMCLK_HCLKX2);
        DrvDPWM_SetSampleRate(SampleRate);
        DrvDPWM_Enable();
}
```

Specify the DPWM parameter in while loop to observe different parameter effect of DPWM.

```
while(!StopTest){
            u32Clk =       DrvSYS_GetHCLK() * 1000;
            u8Freq =       DPWM->CTRL.Freq;

            switch(u8Freq){
              case 0: Div = 228; break;
              case 1: Div = 156; break;
              case 2: Div = 76 ; break;
              case 3: Div = 52 ; break;
              case 4: Div = 780; break;
              case 5: Div = 524; break;
              case 6: Div = 396; break;
              default: Div = 268; break;
            }
```

```
DrvDPWM_DisablePDMA();
DrvADC_PdmaDisable();
DrvDPWM_Close();
DrvADC_Close();
DrvPDMA_Close();
```

```
printf("    DPWM Clk Frequency %d.%d MHz\n", u32Clk/1000000, u32Clk -
((u32Clk/1000000)*1000000));
u32Clk/=Div;
printf("    Carrier Frequency %d.%d kHz\n", u32Clk/1000, u32Clk - ((u32Clk/1000)*1000));
printf("    Dither Level %d\n", DPWM->CTRL.Dither);
printf("    Deatime Level %d\n", DPWM->CTRL.Deadtime);

printf(" c - change DPWM Clk Freq. f - change carrier freq.\n t - Change Deadtime. d - Change dither
\n x - Quit\n\n");
                c=getchar();
                switch(c){
                case 'c':
                        SYSCLK->CLKSEL1.DPWM_S = !SYSCLK->CLKSEL1.DPWM_S;
                    break;
                case 'f':
                        DPWM->CTRL.Freq++;
                    break;
                case 't':
                        DPWM->CTRL.Deadtime++;
                    break;
                case 'd':
                    DPWM->CTRL.Dither++;
                        break;
                case 'x':
                    StopTest=TRUE;
                    break;
                }
}

printf("\n    DPWM sample code is complete.\n\n");
```

ADC received by PDMA interrupt callback function. Switch to another buffer to hold data.

```
void PDMA2_Callback()
{
        extern int32_t IntCnt;
        static uint8_t PingPong=0;
        c_ctrl.Ch1AudioDataRdy[PingPong] = TRUE;
        PingPong ^= 1;
        PDMA->channel[eDRVPDMA_CHANNEL_2].DAR = (uint32_t)&audio_buffer[PingPong+2][0];
        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_2);
};
```

DPWM transmit by PDMA interrupt callback function. Switch to another buffer to transmit data.

```
void PDMA3_Callback(uint32_t status)
{
      static uint8_t PingPong, TxSilence;

      if(status == DPWM_INITIALIZE){
            // Callback can be called from compressor to initialize
            // the buffers.
            PingPong = 0;
            TxSilence =1;
            // Point PDMA to silence buffer
            // Set Source Address
            PDMA->channel[eDRVPDMA_CHANNEL_3].SAR = (uint32_t)&zero_buf;
            PDMA->channel[eDRVPDMA_CHANNEL_3].BCR = ZERO_BUF_SAMPLES *
sizeof(int16_t);
            DrvPDMA_CHEnablelTransfer (eDRVPDMA_CHANNEL_3 );
```

```
}else{
            if(TxSilence){
                // We are waiting for valid data in the output buffer,
                // check to see if present and start transfer
                if(c_ctrl.Ch1AudioDataRdy[PingPong] == TRUE){
                        PDMA->channel[eDRVPDMA_CHANNEL_3].SAR =
(uint32_t)&audio_buffer[PingPong+2][0];
                    PDMA->channel[eDRVPDMA_CHANNEL_3].BCR = MAX_FRAMESIZE *
sizeof(int16_t);
                      TxSilence =0;
                }
            }else{
                // Getting here means we are finished with the current buffer
                // of audio data. Can tell compressor it can process the next
                // one.
                c_ctrl.Ch1AudioDataRdy[PingPong] = FALSE;
                PingPong ^= 1;
                if(c_ctrl.Ch1AudioDataRdy[PingPong] == FALSE){
                    TxSilence =1;
                    // Point PDMA to silence buffer
                    // Set Source Address
                    PDMA->channel[eDRVPDMA_CHANNEL_3].SAR = (uint32_t)&zero_buf;
                   PDMA->channel[eDRVPDMA_CHANNEL_3].BCR = ZERO_BUF_SAMPLES *
sizeof(int16_t);
                }else{
                    // Data ready
                    PDMA->channel[eDRVPDMA_CHANNEL_3].SAR =
(uint32_t)&audio_buffer[PingPong+2][0];
                    PDMA->channel[eDRVPDMA_CHANNEL_3].BCR = MAX_FRAMESIZE *
sizeof(int16_t);
                      TxSilence =0;
                }

            }// if(TxSilence) else
            DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_3);
      } // if(status == I2S_INITIALIZE) else
}
```

DPWM_Play set up the PDMA channel for ADC and DPWM.

```
void DPWM_Play(void)
{
    STR_PDMA_T sPDMA;

    // PDMA Init
   DrvPDMA_Init();

    // PDMA Setting
    PDMA_GCR->PDSSR.DPWM_TXSEL = eDRVPDMA_CHANNEL_3;
   //PDMA_GCR->PDSSR.I2S_RXSEL = eDRVPDMA_CHANNEL_2;
    PDMA_GCR->PDSSR.ADC_RXSEL = eDRVPDMA_CHANNEL_2;

    // CH3 DPWM TX Setting
    sPDMA.sSrcAddr.u32Addr          = (uint32_t)&zero_buf;
   sPDMA.sDestAddr.u32Addr          = (uint32_t)&DPWM->FIFO;
   sPDMA.u8TransWidth               = eDRVPDMA_WIDTH_16BITS;
    sPDMA.u8Mode                    = eDRVPDMA_MODE_MEM2APB;
    sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_INCREMENTED;
    sPDMA.sDestAddr.eAddrDirection  = eDRVPDMA_DIRECTION_FIXED;
    sPDMA.i32ByteCnt                = ZERO_BUF_SAMPLES * 4;
    //sPDMA.i32ByteCnt = MAX_FRAMESIZE * sizeof(int16_t);
    DrvPDMA_Open(eDRVPDMA_CHANNEL_3,&sPDMA);

    // CH2 IIS RX Setting
    //sPDMA.sSrcAddr.u32Addr          = (uint32_t)&I2S->RXFIFO;
    // CH2 ADC RX Setting
    sPDMA.sSrcAddr.u32Addr          = (uint32_t)&SDADC->ADCOUT;
   sPDMA.sDestAddr.u32Addr          = (uint32_t)&audio_buffer[2][0];
    sPDMA.u8TransWidth              = eDRVPDMA_WIDTH_16BITS;
    sPDMA.u8Mode                    = eDRVPDMA_MODE_APB2MEM;
    sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_FIXED;
    sPDMA.sDestAddr.eAddrDirection  = eDRVPDMA_DIRECTION_INCREMENTED;
   sPDMA.i32ByteCnt = MAX_FRAMESIZE * sizeof(int16_t);
   DrvPDMA_Open(eDRVPDMA_CHANNEL_2,&sPDMA);
```

```
// Enable INT
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_3, eDRVPDMA_BLKD );
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_2, eDRVPDMA_BLKD );

        // Install Callback function
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_3, eDRVPDMA_BLKD,
(PFN_DRVPDMA_CALLBACK) PDMA3_Callback );
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_3, eDRVPDMA_TABORT,
(PFN_DRVPDMA_CALLBACK) PDMA3_TA_Callback );
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_2, eDRVPDMA_BLKD,
(PFN_DRVPDMA_CALLBACK) PDMA2_Callback );

        // Enable I2S PDMA and Trigger PDMA specified Channel
       //DrvI2S_EnableRxDMA (TRUE);
       //DrvI2S_EnableRx(TRUE);

        // Enable ADC PDMA and Trigger PDMA specified Channel
        DrvADC_PdmaEnable();
        DrvDPWM_EnablePDMA();

        //IntCnt = 0;
        //IsTestOver=FALSE;

        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_2);
        //DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_3);
        PDMA3_Callback(       DPWM_INITIALIZE );
}
```

5. Execution Environment Setup and Result

- Prepare ISD9160 EVB
- Connect microphone input to ISD9160 by proper jump.
- Connect UART to user's host UART port by female to female cable.
- Compile it under Keil environment.
- Choose proper selection and the result will show on host console window.

6. Revision History

Version	Date	Description
V1.00.01	Sep. 2011	Created