ADC 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.

Table of

1. Introduction

The ISD91XX series includes a 2nd Order Delta-Sigma Audio Analog-to-Digital converter providing SNR >85dB and THD >70dB. The converter can run at sampling rates up to 6.144MHz while a configurable decimation filter allows over sampling ratios of 64/128/192 and 384. This provides support for standard audio sampling rates from 8kHz to 48kHz. The Smpl_DrvADC.c demos the usage of ISD9160 ADC IP.

1.1 Feature

1. Using 16K sampling rate to demo.
2. Demo Microphone to SPK.
3. Demo ADC to IIS.
4. Demo single mode and cycle mode.
5. Demo monitor mode.
6. Demo comparator.
7. Interrupt method.

1.2 Limitation

1. Front-end PGA providing gain range of -12dB – 51dB.
2. Input Boost gain supports 0dB or 26dB.
3. Support sampling rates are 8K, 16K 24K, 48K Hz.
4. Maximum clock rate of Delta-Sigma Converter is 6.144MHz.
5. Audio data buffered to 8 words FIFO, accessible via APB and PDMA.

2. Block Diagram

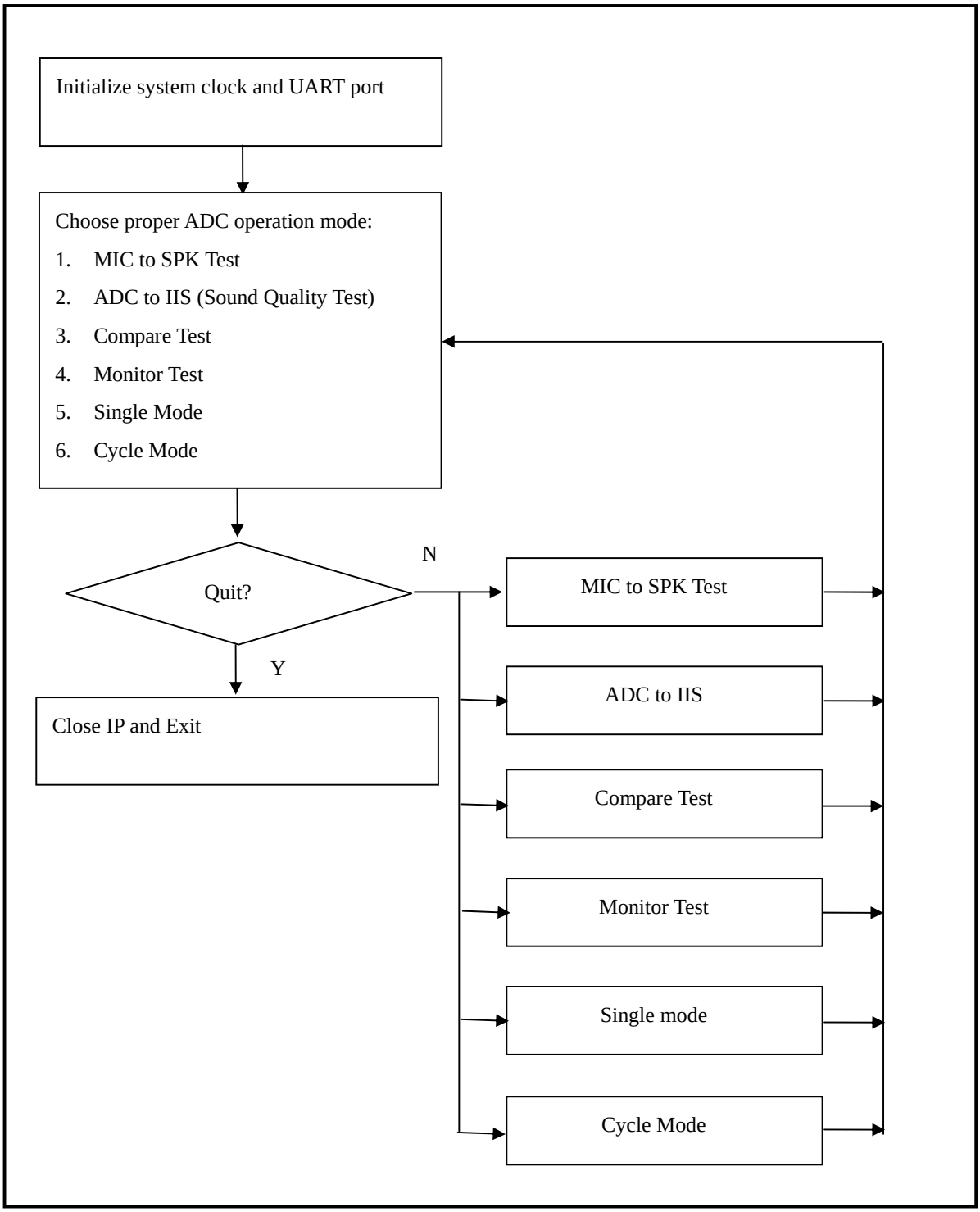

Figure 2-1: Main flow of sample code

3. Calling Sequence

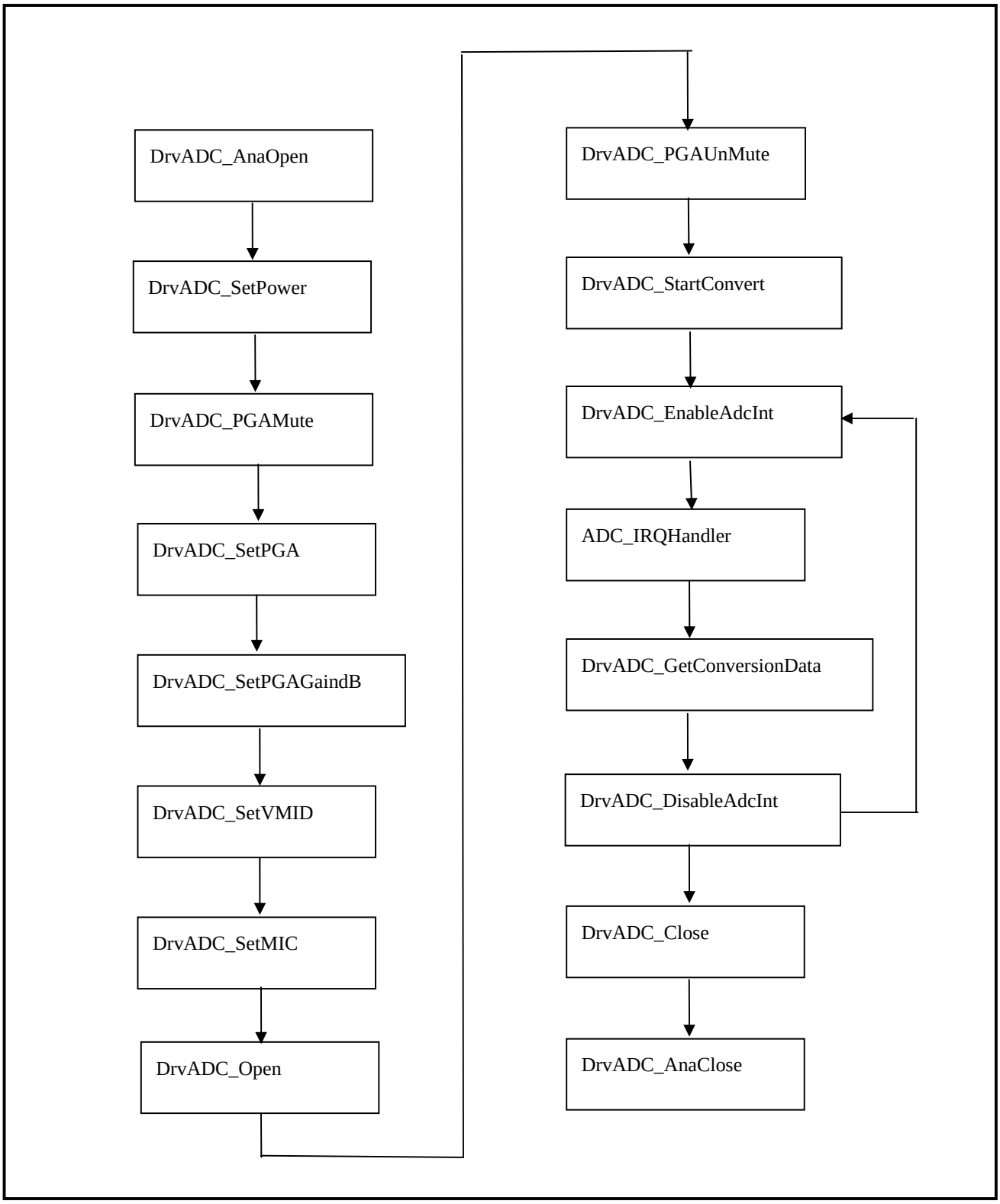

Figure 3-1: API call flow

3.1 API Usage Reference

- ADC Driver User Guide.pdf

4. Code Section

4.1 Constant and Variable

There are 3 interrupt flags “gu8AdcIntFlag, gu8AdcCmp0IntFlag, gu8AdcCmp1IntFlag ”used in interrupt callback function. 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.

```
uint8_t gu8AdcIntFlag;
uint8_t gu8AdcCmp0IntFlag;
uint8_t gu8AdcCmp1IntFlag;

#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};
```

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();

printf("\n\n");
printf("This code tests the ADC function for ISD9160 Chip.\n");
printf("ADC Driver version: %x\n", DrvADC_GetVersion());
```

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 selection menu.

```
printf("\n\n\n");
printf("+------------------------------------------------------------------------+\n");
printf("|                        ADC Test code                                   |\n");
printf("+------------------------------------------------------------------------+\n");
printf("|    [1] MIC To SPK Test                                                 |\n");
printf("|    [2] ADC Sound Quality Test                                          |\n");
printf("|    [3] ADC Compare Monitor Test                                        |\n");
printf("|    [4] ADC Temperature Monitor Test                                    |\n");
printf("|    [5] VMID Test                                                       |\n");
printf("|    [6] MICBIAS Test                                                    |\n");
printf("|    [7] ADC Single Mode Test                                            |\n");
printf("|    [8] ADC Cycle Mode Test                                             |\n");
printf("|    [9] NMI Test                                                        |\n");
printf("|    [q] Quit                                                            |\n");
printf("+------------------------------------------------------------------------+\n");
printf("    Select the test number 1~9 or q:");
```

Select demo function

```
u8Option = getchar();
if(u8Option == '1')
{
    AdcByPDMATest();
}
else if(u8Option == '2')
{
    AdcSoundQualityTest();
}
else if(u8Option == '3')
{
    AdcCompMonitorTest();
}
else if(u8Option == '4')
{
    AdcTemperatureMonitorTest();
}
else if(u8Option == '5')
{
    VMIDTest();
}
else if(u8Option == '6')
{
    MICBIASTest();
}
else if(u8Option == '7')
{
    AdcSingleModeTest();
}
else if(u8Option == '8')
{
    AdcCycleModeTest();
}
```

If user wants to quit, press ‘q’ or ‘Q’ to exit.

```
else if(u8Option == '9')
{
    NMITest();
}
else if( (u8Option == 'q') || (u8Option == 'Q') )
{
    printf("\nADC sample code exit.\n");
    break;
}
```

4.3 Mic to SPK Test

This is a feed through test. Voice is recorded from microphone and play to speaker immediately. User call InitialADC() firstly to setup ADC initial configuration: microphone input, sampling rate is 16KHz. Boost gain is 0dB. PGA gain is 0dB. The ADC data is moved to audio_buffer[2][0] by PDMA transfer. DrvPDMA_Init() enable PDMA clock. DrvPDMA_Open() opens PDMA IP for ADC to audio buffer path.

```
void AdcByPDMATest(void)
{
    STR_PDMA_T sPDMA;
    uint8_t u8Option;
    int32_t i32PGAGaindB = 0; //dB
    int32_t i32PGAGainSet;
    printf("\n=== MIC To SPK test ===\n");
    InitialADC();
    //------------------------------------------------------------------------
    // PDMA Init
    DrvPDMA_Init();
    // CH2 ADC RX Setting
    sPDMA.sSrcAddr.u32Addr          = (uint32_t)&SDADC->ADCOUT;
    sPDMA.sDestAddr.u32Addr         = (uint32_t)&audio_buffer[2][0];
    sPDMA.u8Mode                    = eDRVPDMA_MODE_APB2MEM;
    sPDMA.u8TransWidth              = eDRVPDMA_WIDTH_16BITS;
    sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_FIXED;
    sPDMA.sDestAddr.eAddrDirection = eDRVPDMA_DIRECTION_INCREMENTED;
    sPDMA.i32ByteCnt = MAX_FRAMESIZE * sizeof(int16_t);
    DrvPDMA_Open(eDRVPDMA_CHANNEL_0, &sPDMA);
```

DrvPDMA_SetCHForAPBDevice set up ADC to use PDMA channel 0 for read from ADC. DrvPDMA_EnableInt enables channel 0 block transfer end interrupt. DrvPDMA_InstallCallback installs callback function PDMA0_Callback for PDMA IRQ handler. DrvADC_PdmaEnable turns on ADC PDMA receive mechanism. The initialDPWM(16000) initialize DPWM block for 16KHz sampling rate operation and start to driver PWM signal on SPK+ and SPK-. DPWM_Play initialize PDMA transmitting setting for DPWM. DrvADC_StartConvert let ADC circuit start convert analog signal to digital data and put in FIFO. DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0) lets PDMA channel 0 start to work for ADC to move FIFO data to audio_buffer.

```
        // PDMA Setting
        DrvPDMA_SetCHForAPBDevice(
        eDRVPDMA_CHANNEL_0,
        eDRVPDMA_ADC,
        eDRVPDMA_READ_APB
        );

        // Enable INT
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD );

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

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

        // Enable DPWM and set sampling rate
        InitialDPWM(16000);
        //InitialDPWM(8000);
        DPWM_Play();

        // Start A/D conversion
        DrvADC_StartConvert();

        // start ADC PDMA transfer
        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
```

This while loop sets up the PGA gain for ADC. If user input a ‘q’, this Test will close PDMA, DPWM, and ADC and exit.

```
        while(1)
        {
                printf("\nChange ADC parameter\n");
                printf("    [i] increase PGA gain\n");
                printf("    [d] decrease PGA gain\n");
                printf("    [q] Exit\n");
                u8Option = getchar();

                if(u8Option=='i') {
                        i32PGAGaindB += 50;
                        i32PGAGainSet=DrvADC_SetPGAGaindB(i32PGAGaindB);
                        printf("Current PGA Gain = %c%d.%d dB\n", sign(i32PGAGainSet),
abs(i32PGAGainSet)/100, abs(i32PGAGainSet)%100);
                }
                else if(u8Option=='d') {
                        i32PGAGaindB -= 50;
                        i32PGAGainSet=DrvADC_SetPGAGaindB(i32PGAGaindB);
                        printf("Current PGA Gain = %c%d.%d dB\n", sign(i32PGAGainSet),
abs(i32PGAGainSet)/100, abs(i32PGAGainSet)%100);
                }
                else if(u8Option=='q')
                        break;
        }

        DrvPDMA_Close();
        DrvADC_Close();
        DrvDPWM_Close();
}
```

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

```
void InitialADC(void)
{
    S_DRVADC_PARAM sParam;
    uint32_t u32AdcStatus;
    uint32_t OSR;

    // b0,b1,b2,a1,a2,b0,b1,b2,a1,a2,b0,b1,b2,a1,a2
    /*
    uint32_t u32BiqCoeff[15]={0x10000, 0x15b8a, 0x10000, 0x15068, 0x0ef98,
                              0x10000, 0x00000,  0x00000, 0x00000, 0x00000,
                              0x10000, 0x00000,  0x00000, 0x00000, 0x00000};
    */

    /* 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_OFF);

    /* 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);

/* ALC Setting */
//ALC->ALC_CTRL.ALCLVL = 15;
//ALC->ALC_CTRL.ALCSEL =1;
//ALC->ALC_CTRL.ALCSEL =0;
//ALC->ALC_CTRL.ALCDCY = 3;
//ALC->ALC_CTRL.NGEN =1;

/* Open ADC block */
sParam.u8AdcDivisor      = 0;
sParam.u8SDAdcDivisor = 16;              //OSR192 :32 for 8K, 16 for 16K
sParam.eOSR                  = eDRVADC_OSR_192;
sParam.eInputSrc      = eDRVADC_MIC;
sParam.eInputMode = eDRVADC_DIFFERENTIAL;
sParam.u8ADCFifoIntLevel = 7;
u32AdcStatus=DrvADC_Open(&sParam);
if(u32AdcStatus == E_SUCCESS) {
        printf("ADC has been successfully opened.\n");
        printf("ADC clock divisor=%d\n",SYSCLK->CLKDIV.ADC_N);
        printf("ADC over sampling clock divisor=%d\n",SDADC->CLK_DIV);
        switch(SDADC->DEC.OSR)
        {
           case eDRVADC_OSR_64:OSR=64;break;
           case eDRVADC_OSR_128:OSR=128;break;
           case eDRVADC_OSR_192:OSR=192;break;
```

```
                case eDRVADC_OSR_384:OSR=384;break;
            }
            printf("ADC over sampling ratio=%d\n", OSR);
            printf("Select microphone path as differential input\n");
            printf("ADC Fifo Interrupt Level=%d\n", SDADC->INT.FIFO_IE_LEV);
            printf("Conversion rate: %d samples/second\n", DrvADC_GetConversionRate());
        }
        else {
            printf("ADC Open failed!\n");
        }
        /* Change Decimation and FIFO Setting */
        //DrvADC_SetAdcOverSamplingClockDivisor(u8SDAdcDivisor);
        //DrvADC_SetOverSamplingRatio(eOSR);
        //DrvADC_SetCICGain(u8CICGain);
        //DrvADC_SetFIFOIntLevel(u8ADCFifoIntLevel);

        /* Change BIQ Setting */
        //SYSCLK->APBCLK.BIQ_EN = 1;
        //SYS->IPRSTC2.BIQ_RST = 1;
        //SYS->IPRSTC2.BIQ_RST = 0;
        //DrvADC_SetBIQ(1023, 1, eDRVADC_BIQ_IN_ADC, u32BiqCoeff);
        /* Interrupt Setting */
        //DrvADC_EnableAdcInt(DRVADC_ADC_CALLBACK Callback, uint32_t u32UserData);
        DrvADC_PGAUnMute(eDRVADC_MUTE_PGA);
}
```

InitialDPWM open DPWM block and set the sampling rate.

```
void InitialDPWM(uint32_t u32SampleRate)
{
        DrvDPWM_Open();
        DrvDPWM_SetDPWMClk(E_DRVDPWM_DPWMCLK_HCLKX2);
        //DrvDPWM_SetDPWMClk(E_DRVDPWM_DPWMCLK_HCLK);
        DrvDPWM_SetSampleRate(u32SampleRate);
        DrvDPWM_Enable();
}
```

To use DPWM, PDMA channel is set up for the DPWM. DrvPDMA_Open specifies channel 1 transfer parameter. DrvPDMA_SetCHForAPBDevice links channel 1 to DPWM for write. DrvPDMA_EnableInt enable channel interrupt for block transfer end. DrvPDMA_InstallCallback install PDMA1_Callback for PDMA IRQ Handler of block transfer end and PDMA1_TA_Callback for PDMA IRQ Handler of transfer abort. DrvDPWM_EnablePDMA enable DPWM PDMA function. PDMA1_Callback(I2S_INITIALIZE) will start PDMA for first transfer.

```
void DPWM_Play(void)
{
        STR_PDMA_T sPDMA;
        // CH1 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;
        DrvPDMA_Open(eDRVPDMA_CHANNEL_1,&sPDMA);
        // PDMA Setting
        DrvPDMA_SetCHForAPBDevice(
        eDRVPDMA_CHANNEL_1,
        eDRVPDMA_DPWM,
        eDRVPDMA_WRITE_APB
        );
        // Enable INT
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD );
        // Install Callback function
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD,
(PFN_DRVPDMA_CALLBACK) PDMA1_Callback );
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_1, eDRVPDMA_TABORT,
(PFN_DRVPDMA_CALLBACK) PDMA1_TA_Callback );

        // Enable DPWM PDMA
        DrvDPWM_EnablePDMA();
        PDMA1_Callback(I2S_INITIALIZE );
}
```

4.4 ADC to I2S (ADC Sound Quality Test)

This test demonstrates record voice from ADC and transmits to I2S interface. It can be used to test the ADC performance by Audio Precision (an audio test instrument). Firstly, InitialADC initialize ADC for microphone path. The PDMA is used to transfer data. Call DrvPDMA_Open, DrvPDMA_SetCHForAPBDevice to set up PDMA. Here channel 0 was assigned to do PDMA transfer.

```
void AdcSoundQualityTest()
{
    STR_PDMA_T sPDMA;
    uint8_t u8Option;
    int32_t i32PGAGainSet;
    int32_t i32PGAGaindB = 0; //dB
    printf("\n=== ADC Sound Quality test ===\n");
    InitialADC();
    //------------------------------------------------------------------
    // PDMA Init
    DrvPDMA_Init();
    // CH2 ADC RX Setting
    sPDMA.sSrcAddr.u32Addr          = (uint32_t)&SDADC->ADCOUT;
    sPDMA.sDestAddr.u32Addr         = (uint32_t)&audio_buffer[2][0];
    sPDMA.u8Mode                    = eDRVPDMA_MODE_APB2MEM;
    sPDMA.u8TransWidth              = eDRVPDMA_WIDTH_16BITS;
    sPDMA.sSrcAddr.eAddrDirection   = eDRVPDMA_DIRECTION_FIXED;
    sPDMA.sDestAddr.eAddrDirection= eDRVPDMA_DIRECTION_INCREMENTED;
    sPDMA.i32ByteCnt = MAX_FRAMESIZE * sizeof(int16_t);
    DrvPDMA_Open(eDRVPDMA_CHANNEL_0, &sPDMA);
    // PDMA Setting
    DrvPDMA_SetCHForAPBDevice(
    eDRVPDMA_CHANNEL_0,
    eDRVPDMA_ADC,
    eDRVPDMA_READ_APB
    );
```

The DrvPDMA_EnableInt and DrvPDMA_InstallCallback set up the interrupt callback function and enable the interrupt. DrvADC_PdmaEnable turn on the PDMA request for ADC. DrvADC_Startconvert starts to convert data to ADC FIFO. For transmit path, InitialI2S initialize I2S interface for 16KHz sampling rate. I2S_Play set up PDMA parameter for I2S. When all set up are ready, DrvPDMA_CHEnableTransfer start PDMA transfer from ADC FIFO to audio buffer.

```
    // Enable INT
    DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD );
    // Install Callback function
    DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_0, eDRVPDMA_BLKD,
(PFN_DRVPDMA_CALLBACK) PDMA0_Callback );

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

    // I2S interface
    InitialI2S(16000);
    I2S_Play();

    // Start A/D conversion
    DrvADC_StartConvert();

    // start ADC PDMA transfer
    DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
```

This while loop sets up the PGA gain for ADC. If user input a ‘q’, this Test will close PDMA, I2S, and ADC and exit.

```
        while(1)
        {
                printf("ADC record and send to I2S interface\n");
                printf("    [i] increase PGA gain\n");
                printf("    [d] decrease PGA gain\n");
                printf("    [q] Exit\n");
                u8Option = getchar();

                if(u8Option=='i') {
                        i32PGAGaindB += 50;
                        i32PGAGainSet=DrvADC_SetPGAGaindB(i32PGAGaindB);
                        printf("Current PGA Gain = %c%d.%d dB\n", sign(i32PGAGainSet),
abs(i32PGAGainSet)/100, abs(i32PGAGainSet)%100);
                }
                else if(u8Option=='d') {
                       i32PGAGaindB -= 50;
                        i32PGAGainSet=DrvADC_SetPGAGaindB(i32PGAGaindB);

                        printf("Current PGA Gain = %c%d.%d dB\n", sign(i32PGAGainSet),
abs(i32PGAGainSet)/100, abs(i32PGAGainSet)%100);
                }
                else

                if(u8Option=='q')
                        break;
        }
        DrvPDMA_Close();
        DrvADC_Close();
        DrvI2S_Close();
}
```

InitialI2S set up I2S interface. Firstly, configure SYSCLK->CLKSE2_I2S_S to select I2S clock source. User fills a structure variable st and pass it to DrvI2S_Open to setup parameter. Here 16bit, stereo, I2S format and master mode are selected. The FIFO level is choosing to 4. I2S->CON.MCLKEN enable the MCLK. SYS->GPA_ALT.GPA0=2 will route MCLK to GPA0.

```
void InitialI2S(uint32_t u32SampleRate)
{
        S_DRVI2S_DATA_T st;
        SYSCLK->CLKSEL2.I2S_S=3;
        /* Set I2S Parameter */
        switch(u32SampleRate)
        {
          case 8000:    st.u32SampleRate      = 8000; break;
          case 16000: st.u32SampleRate      = 16000; break;
          case 48000: st.u32SampleRate      = 48000; break;
          default:      st.u32SampleRate      = 16000; break;
        }
        st.u8WordWidth          = DRVI2S_DATABIT_16;
        st.u8AudioFormat        = DRVI2S_STEREO;
        st.u8DataFormat         = DRVI2S_FORMAT_I2S;
        st.u8Mode               = DRVI2S_MODE_MASTER;
        st.u8RxFIFOThreshold = DRVI2S_FIFO_LEVEL_WORD_4;
        st.u8TxFIFOThreshold = DRVI2S_FIFO_LEVEL_WORD_4;
        DrvI2S_Open(&st);
        I2S->CON.MCLKEN = 1;
        SYS->GPA_ALT.GPA0 = 2;    // MCLK Output
        switch(u32SampleRate)
        {
          case 8000:    I2S->CLKDIV.BCLK_DIV = 95; break;
          case 16000: I2S->CLKDIV.BCLK_DIV = 47; break;
          case 48000: I2S->CLKDIV.BCLK_DIV = 15; break;
          default:      I2S->CLKDIV.BCLK_DIV = 47; break;
        }

        I2S->CLKDIV.MCLK_DIV = 6 ;     // 6
```

Select GPA alternate function for I2S interface and Enable I2S for receiving and transmiting.

```
    /* Set I2S I/O */
    //DrvGPIO_InitFunction(FUNC_I2S0);
      SYS->GPA_ALT.GPA4              =1;     //
      SYS->GPA_ALT.GPA5              =1;     //
      SYS->GPA_ALT.GPA6              =1;     //
      SYS->GPA_ALT.GPA7              =1;     //

      /* Enable I2S Tx/Rx function */
      DrvI2S_EnableRx(TRUE);
      DrvI2S_EnableTx(TRUE);
}
```

Prepare PDMA channel 1 for buffer to I2S transfer.

```
void I2S_Play(void)
{
      STR_PDMA_T sPDMA;

      // CH1 I2S TX Setting
      sPDMA.sSrcAddr.u32Addr          = (uint32_t)&zero_buf;
      sPDMA.sDestAddr.u32Addr         = (uint32_t)&(I2S->TXFIFO);
      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;
      DrvPDMA_Open(eDRVPDMA_CHANNEL_1, &sPDMA);
```

Link channel 1 for I2S. PDMA will transfer data to I2S. DrvPDMA_InstallCallBack will install PDMA1_Callback function to PDMA IRQ Handler for block transfer end and PDMA1_TA_Callback function to PDMA IRQ Handler for transfer abort. DrvPDMA_EnableInt enables the interrupt. DrvI2S_EnableTxDMA(TRUE) and DrvI2SEnableTx(TRUE) turn on PDMA request for I2S. PDMA1_Callback(I2S_INITIALIZE) starts first PDMA transfer for I2S.

```
        // PDMA Setting
        DrvPDMA_SetCHForAPBDevice(
        eDRVPDMA_CHANNEL_1,
        eDRVPDMA_I2S,
        eDRVPDMA_WRITE_APB
        );

        // Install Callback function
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD,
(PFN_DRVPDMA_CALLBACK) PDMA1_Callback );
        DrvPDMA_InstallCallBack(eDRVPDMA_CHANNEL_1, eDRVPDMA_TABORT,
(PFN_DRVPDMA_CALLBACK) PDMA1_TA_Callback );

        // Enable INT
        DrvPDMA_EnableInt(eDRVPDMA_CHANNEL_1, eDRVPDMA_BLKD );

        // Enable I2S PDMA
        DrvI2S_EnableTxDMA (TRUE);
        DrvI2S_EnableTx(TRUE);

        PDMA1_Callback(       I2S_INITIALIZE );
}
```

4.5 ADC Compare Monitor Test

This test demonstrates the digital comparator usage in ADC path. The menu here requires the user to input a threshold option.

```
void AdcCompMonitorTest()
{
    uint8_t u8CmpChannelNum, u8CmpMatchCount;
    uint32_t u32ConversionData[8];
    uint8_t i;
    uint32_t u32HiTh,u32LoTh;
    uint8_t Option;
    u32HiTh     = 0x7000;
    u32LoTh     = 0x9000;
    show_menu:
    printf("\n\n");
    printf("+------------------------------------------------------------------------+\n");
    printf("|                    ADC compare monitor test                            |\n");
    printf("+------------------------------------------------------------------------+\n");
    printf("Threshold values\n");
    printf("a) 0x7000\n");
    printf("b) 0x6000\n");
    printf("c) 0x5000\n");
    printf("d) 0x4000\n");
    printf("e) 0x3000\n");
    printf("f) 0x2000\n");
    printf("g) 0x1000\n");
    printf("h) 0x0000\n");
    printf("i) 0xf000\n");
    printf("j) 0xe000\n");
    printf("k) 0xd000\n");
    printf("l) 0xc000\n");
    printf("m) 0xb000\n");
    printf("n) 0xa000\n");
    printf("o) 0x9000\n");
    printf("input choice:");
    Option=getchar();
```

Based on the option , set the threshold.

```
Option=getchar();
if(Option == 'a')
        u32LoTh = 0x7000;
else if (Option == 'b')
        u32LoTh = 0x6000;
else if (Option == 'c')
        u32LoTh = 0x5000;
else if (Option == 'd')
        u32LoTh = 0x4000;
else if (Option == 'e')
        u32LoTh = 0x3000;
else if (Option == 'f')
        u32LoTh = 0x2000;
else if (Option == 'g')
        u32LoTh = 0x1000;
else if (Option == 'h')
        u32LoTh = 0x0000;
else if (Option == 'i')
        u32LoTh = 0xf000;
else if (Option == 'j')
        u32LoTh = 0xe000;
else if (Option == 'k')
        u32LoTh = 0xd000;
else if (Option == 'l')
        u32LoTh = 0xc000;
else if (Option == 'm')
        u32LoTh = 0xb000;
else if (Option == 'n')
        u32LoTh = 0xa000;
else if (Option == 'o')
        u32LoTh = 0x9000;
else {
        printf("Invalid option!\n");
```

In the sample code, u32HiTh is set to u32LoTh. User may change it by your application purpose. Call InitialADC to initialize ADC. The default input is microphone. If user wants to use another channel, he/she may call DrvADC_SetAdcChannel to select another input channel. The input channel can be single mode for GPB0~GPB7 or differential mode for GPB01, GPB23, GPB45 and GPB67. Set the convert length for one interrupt by DrvADC_SetFIFOIntLevel. Ignore first converted data. The first interrupt data is got when ADC start to convert from idle state to work state. The value is not reliable so discard them. Ignore second interrupt data, the purpose is the same with first interrupt to make sure that in the following interrupt the converted data correctly responds to analog input.

```
        goto show_menu;
    }
    u32HiTh = u32LoTh;
    printf("Test Lo threshold = %x Hi threshold = %x\n", u32LoTh, u32HiTh);
    InitialADC();
    //use GPB0/1 for test
    //DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH1_IN_N);
    //DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_DIFFERENTIAL_CH01);

    DrvADC_SetFIFOIntLevel(7);
    //------------------------------------------------
    DrvADC_EnableAdcInt(AdcIntCallback, 0);
    // start A/D conversion
    DrvADC_StartConvert();
    while(gu8AdcIntFlag==0);
            gu8AdcIntFlag = 0;
    for(i=1;i<=8;i++)
        DrvADC_GetConversionData();

    DrvADC_EnableAdcInt(AdcIntCallback, 0);
    while(gu8AdcIntFlag==0);
            gu8AdcIntFlag = 0;
    for(i=1;i<=8;i++)
        DrvADC_GetConversionData();
    //-------------------------------------------------------------------------
    u8CmpChannelNum = 0;
    gu8AdcCmp0IntFlag = 0;
    u8CmpMatchCount = 8;
```

Here the code uses greater comparator. Call DrvADC_Adcmp1Enable, specify the required operation, threshold and compare match count. Compare match count specify the compare condition must hold for u8CmpMatchCount then the interrupt occurs. Enable ADC compare 1 interrupt and set the callback function by DrvADC_EnableAdcmp1Int. In the while loop, continue to read ADC data till compare condition holds.

```
// Enable ADC compare 0. Compare condition: conversion result < 0x9000.
//DrvADC_Adcmp0Enable(eDRVADC_LESS_THAN, u32HiTh, u8CmpMatchCount);
//DrvADC_Adcmp0Enable(eDRVADC_LESS_THAN, u32LoTh, u8CmpMatchCount);

// enable ADC compare 0 interrupt and set the callback function
//DrvADC_EnableAdcmp0Int(Cmp0IntCallback, 0);

//------------------------------------------------------------------------
gu8AdcCmp1IntFlag = 0;
u8CmpMatchCount = 8;
// Enable ADC compare 1. Compare condition: conversion result >= 0x7000.
DrvADC_Adcmp1Enable(eDRVADC_GREATER_OR_EQUAL, u32HiTh,    u8CmpMatchCount);
//DrvADC_Adcmp1Enable(eDRVADC_GREATER_OR_EQUAL, u32LoTh,    u8CmpMatchCount);

// enable ADC compare 1 interrupt and set the callback function
DrvADC_EnableAdcmp1Int(Cmp1IntCallback, 0);

// Wait ADC compare interrupt
while( (gu8AdcCmp1IntFlag==0) )    {
                DrvADC_EnableAdcInt(AdcIntCallback, 0);
                while(gu8AdcIntFlag==0);
                gu8AdcIntFlag = 0;
                //printf("--\n");
                for(i=0;i<=7;i++) {
                        u32ConversionData[i] = DrvADC_GetConversionData();
                }
                for(i=0;i<=7;i++) {
                        printf("0x%X (%d)\n", u32ConversionData[i], u32ConversionData[i]);
                }
                SysTimerDelay(300000);
}
```

When compare interrupt hold, leave the while loop. Dump the converted data again. Call DrvADC_DisableAdcmp0Int and DrvADC_DisableAdcmp1Int to disable interrupt. Call DrvADC_Adcmp0Disable and DrvADC_Adcmp1Disable to disable comparator function. Base on interrupt flag, print the message. Call DrvADC_StopConvert if user wants to stop ADC conversion.

```
                printf("--\n");
                for(i=0;i<=7;i++) {
                        u32ConversionData[i] = DrvADC_GetConversionData();
                }
                for(i=0;i<=7;i++) {
                        printf("0x%X (%d)\n", u32ConversionData[i], u32ConversionData[i]);
                }
        //DrvADC_StopConvert();
        DrvADC_DisableAdcmp0Int();
        DrvADC_DisableAdcmp1Int();
        DrvADC_Adcmp0Disable();
        DrvADC_Adcmp1Disable();

        if(gu8AdcCmp0IntFlag==1)
        {
        printf("The conversion result of channel %d is less than 0x%X\n", u8CmpChannelNum, u32LoTh);
        }
        else
        {
        printf("The conversion result of channel %d is greater or equal to 0x%X\n", u8CmpChannelNum, u32HiTh);
        }
}
```

4.6 ADC Temperature Monitor Test

This function demos the ADC temperature monitor usage. Firstly, InitialADC will initial ADC clock sample rate, FIFO level, and analog input to microphone. To use temperature monitor, call DrvADC_SetAdcChannel and pass eDRVADC_TEMP, eDRVADC_DIFFERENTIAL. This will switch analog input for temperature sensing. DrvADC_SetPGA turn on the power of PGA, BOOST stage and select VBG as reference voltage. The DrvADC_SetPGAGaindB(525) will scale input by 5.25dB. DrvADC_SetFIFOIntLevel(1) lets interrupt occur when every sample converted. SrvADC_StartConvert start converting data. The for loop is used to flush the initial FIFO data in ADC.

```
void AdcTemperatureMonitorTest(void)
{
    uint8_t u8Option;
    uint8_t i;
    uint32_t u32ConversionData;
    int32_t i32Temp;
    printf("\n=== ADC Temperature mode test ===\n");
    InitialADC();
    // For Temperature setting
    DrvADC_SetAdcChannel(eDRVADC_TEMP, eDRVADC_DIFFERENTIAL);
    DrvADC_SetPGA(
        eDRVADC_REF_SEL_VBG,
        eDRVADC_PU_PGA_ON,
        eDRVADC_PU_BOOST_ON,
        eDRVADC_BOOSTGAIN_0DB);
    DrvADC_SetPGAGaindB(525);
    DrvADC_SetFIFOIntLevel(1);          // interrupt occurs when one sample is converted.
    // Start A/D conversion
    DrvADC_StartConvert();
    for(i=1;i<=10;i++) {
    // Enable ADC Interrupt function
        DrvADC_EnableAdcInt(AdcIntCallback, 0);
        while(gu8AdcIntFlag==0);
            gu8AdcIntFlag = 0;
        u32ConversionData = DrvADC_GetConversionData();
    }
```

The while loop continuous to convert ADC data and calculates corresponding temperature value. When DrvADC_EnableAdcInt is call, the interrupt is enabled. Owning to achieving the single scan, there is a DrvADC_DisableInt in the ADC callback function to hold the int. So used DrvADC_EnableAdcInt to turn on it again. In sigma-delta ADC, once DrvADC_StartConvert is called, the data is converted automatically in the circuit. User need to control Interrupt enable bit to start or stop to lunch interrupt to get the ADC data.

gu8AdcIntFlag is set when interrupt occurs, judge this flag and call DrvADC_GetConversionData. u32ConversionData hold current ADC data. iTemp is calculate from u32ConversionData by the formula reference from TRM.

```
    while(1)
    {
        printf("Start to get temperature[y/n]:\n");

        u8Option = getchar();
        if(u8Option=='y')
            ;
        else
            return ;

        // Enable ADC Interrupt function
        DrvADC_EnableAdcInt(AdcIntCallback, 0);
            // Start A/D conversion
        //DrvADC_StartConvert();
        // Wait ADC interrupt
        while(gu8AdcIntFlag==0);

        gu8AdcIntFlag = 0;
                u32ConversionData = DrvADC_GetConversionData();
        i32Temp = 27+ (u32ConversionData - 0x42EA) / 50;      //refer to TRM for Temperature formula
        printf("Temperature result: (%d) degree. AD Convernt Data: (%d)\n\n", i32Temp,
u32ConversionData);
    }
}
```

4.7 ADC Single Mode Test

This function demonstrates ADC single mode. Firstly, InitialADC initialize ADC. Here modify ADC FIFO level by DrvADC_SetFIFOLevel. Specify single or differential mode.

```
void AdcSingleModeTest()
{
    int8_tj;
    uint8_t u8Option;
    uint8_t u8InputMode;
    uint32_t u32ConversionData;

    printf("\n== Single mode test ===\n");
    InitialADC();

    DrvADC_SetFIFOIntLevel(7);
    while(1)
    {
        single_menu:
        printf("Select input mode:\n");
        printf("    [1] Single end input\n");
        printf("    [2] Differential input\n");
        printf("    [q] Exit single mode test\n");
        u8Option = getchar();
        if(u8Option=='1')
            u8InputMode = 1; // single-end
        else if(u8Option=='2')
            u8InputMode = 2; // differential
        else if(u8Option=='q')
            return ;
        else
            goto single_menu;
```

For single mode, call SingleEndInput_ChannelSelect() to specify single mode channel. For differential mode, call DifferentialInput_ChannelSelect() to select differential input. DrvADC_EnableAdcInt enable ADC interrupt. DrvADC_StartConvert start ADC conversion. When interrupt flag is set, use DrvADC_GetConversionData to get the FIFO data.

```
            if(u8InputMode==1)
                SingleEndInput_ChannelSelect();          // Select the active channel
            else
                DifferentialInput_ChannelSelect();  // Select the active channel

            // Enable ADC Interrupt function
            DrvADC_EnableAdcInt(AdcIntCallback, 0);

            // Start A/D conversion
            DrvADC_StartConvert();
            // Wait ADC interrupt

            while(gu8AdcIntFlag==0);
            gu8AdcIntFlag = 0;

            for(j=1;j<=8;j++) {
                u32ConversionData = DrvADC_GetConversionData();
                printf("0x%X (%d)\n\n", u32ConversionData, u32ConversionData);
            }
        }
}
```

Select input channel for single end operation.

```
uint8_t SingleEndInput_ChannelSelect()
{
    uint8_t u8Option;

    printf("Select ADC channel:\n");
    printf("    [0] Channel 0\n");
    printf("    [1] Channel 1\n");
    printf("    [2] Channel 2\n");
    printf("    [3] Channel 3\n");
    printf("    [4] Channel 4\n");
    printf("    [5] Channel 5\n");
    printf("    [6] Channel 6\n");
    printf("    [7] Channel 7\n");
    printf("    Other keys: exit single mode test\n");
    u8Option = getchar();
    if(u8Option=='0')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH0_IN_N);
    else if(u8Option=='1')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH1_IN_N);
    else if(u8Option=='2')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH2_IN_N);
    else if(u8Option=='3')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH3_IN_N);
    else if(u8Option=='4')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH4_IN_N);
    else if(u8Option=='5')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH5_IN_N);
    else if(u8Option=='6')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH6_IN_N);
    else if(u8Option=='7')
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_SINGLE_END_CH7_IN_N);
    else
        return 0xFF;
    u8Option = u8Option - '0';
    return u8Option;      // return the the active channel number
}
```

Select input channel for differential operation.

```
uint8_t DifferentialInput_ChannelSelect()
{
    uint8_t u8Option;

    printf("Select ADC channel:\n");
    printf("    [0] Differential input pair 0(CH0 and 1)\n");
    printf("    [1] Differential input pair 1(CH2 and 3)\n");
    printf("    [2] Differential input pair 2(CH4 and 5)\n");
    printf("    [3] Differential input pair 3(CH6 and 7)\n");
    printf("    Other keys: quit\n");
    u8Option = getchar();
    if(u8Option=='0')
    {
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_DIFFERENTIAL_CH01);
    }
    else if(u8Option=='1')
    {
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_DIFFERENTIAL_CH23);
    }
    else if(u8Option=='2')
    {
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_DIFFERENTIAL_CH45);
    }
    else if(u8Option=='3')
    {
        DrvADC_SetAdcChannel(eDRVADC_GPIO, eDRVADC_DIFFERENTIAL_CH67);
    }
    else
        return 0xFF;
    return u8Option;
}
```

4.8 ADC Cycle Mode Test

This function demonstrates ADC cycle mode. Firstly, InitialADC initialize ADC. Here modify ADC FIFO level by DrvADC_SetFIFOLevel. Specify single or differential mode and select the channel number. For single mode, call SingleEndInput_ChannelSelect() to specify single mode channel. For differential mode, call DifferentialInput_ChannelSelect() to select differential input.

```
void AdcCycleModeTest()
{
    int8_t i,j;
    uint8_t u8Option;
    uint8_t u8InputMode;
    uint32_t u32ConversionData[8];

    printf("\n== Cycle mode test ===\n");
    InitialADC();

    DrvADC_SetFIFOIntLevel(7);
    while(1)
    {
        printf("Select input mode:\n");
        printf("    [1] Single end input\n");
        printf("    [2] Differential input\n");
        printf("    [q] Exit cycle mode test\n");
        u8Option = getchar();
        if(u8Option=='1')
            u8InputMode = 1; // single-end
        else if(u8Option=='2')
            u8InputMode = 2; // differential
        else
            return ;

        if(u8InputMode==1)
            SingleEndInput_ChannelSelect();    // Select the active channel
        else
            DifferentialInput_ChannelSelect();  // Select the active channel
```

DrvADC_EnableAdcInt enable ADC interrupt. DrvADC_StartConvert start ADC conversion. When interrupt flag is set, use DrvADC_GetConversionData to get the FIFO data. The for loop will continue to get adc data. The time period between next fetch is controlled by SysTimerDelay(1000000).

```
            for(i=1;i<=30;i++) {
                // Enable ADC Interrupt function
                DrvADC_EnableAdcInt(AdcIntCallback, 0);

                // Start A/D conversion
                DrvADC_StartConvert();

                // Wait ADC interrupt
                while(gu8AdcIntFlag==0);
                gu8AdcIntFlag = 0;

                printf("--\n");
                for(j=0;j<=7;j++) {
                u32ConversionData[j] = DrvADC_GetConversionData();
                }
                for(j=0;j<=7;j++) {
                printf("0x%X (%d)\n", u32ConversionData[j], u32ConversionData[j]);
                SysTimerDelay(1000000);
                }
            }
        }
}
```

Close ADC function.

```
DrvPDMA_Close();      // close when exit
DrvDPWM_Close();    // close when exit
DrvADC_Close();       // close when exit.
DrvADC_AnaClose(); // close analog part
```

4.9 ADC IRQ Handler

The ADC Interrupt has three sources: FIFO over threshold, comparator 0 and comparator 1. The AdcIntCallback is called when FIFO over threshold. It is installed by DrvADC_EnableAdcInt (AdcIntCallback, 0). In this sample code, gu8AdcIntFlag is set to tell program that ADC received data is over the threshold of FIFO, user should call DrvADC_GetConversionData to get the data from FIFO. Cmp0IntCallback and Cmp1IntCallback are the callback function of comparator. They are install by DrvADC_EnableAdcmp0Int(Cmp0IntCallback, 0) and DrvADC_EnableAdcmp1Int(Cmp1IntCallback, 0). They are called when compare condition hold. In this sample code, gu8AdcCmp0IntFlag and gu8AdcCmp1IntFlag are set to tell program the compare condition is hold.

```
/*---------------------------------------------------------------------------------------------------------*/
/* ADC interrupt callback function                                                                         */
/*---------------------------------------------------------------------------------------------------------*/
void AdcIntCallback(uint32_t u32UserData)
{
    gu8AdcIntFlag = 1;
    DrvADC_DisableAdcInt();     // for single mode test
}
/*---------------------------------------------------------------------------------------------------------*/
/* ADC interrupt callback function                                                                         */
/*---------------------------------------------------------------------------------------------------------*/
void Cmp0IntCallback(uint32_t u32UserData)
{
    gu8AdcCmp0IntFlag = 1;
    GPIOA->DOUT = 0x0;
}
/*---------------------------------------------------------------------------------------------------------*/
/* ADC interrupt callback function                                                                         */
/*---------------------------------------------------------------------------------------------------------*/
void Cmp1IntCallback(uint32_t u32UserData)
{
    gu8AdcCmp1IntFlag = 1;
    GPIOA->DOUT = 0x1;
}
```

PDMA0_Callback runs when PDMA move data from ADC to buffer till MAX_FRAMESIZE. Use PingPong buffer to hold next data. PDMA1_TA_Callback runs when transfer aborts.

```
/*---------------------------------------------------------------------------------------------------------*/
/* ADC RX Callback                                                                                         */
/*---------------------------------------------------------------------------------------------------------*/
void PDMA0_Callback()
{
        static uint8_t PingPong=0;
        c_ctrl.Ch1AudioDataRdy[PingPong] = TRUE;
        PingPong ^= 1;
        PDMA->channel[eDRVPDMA_CHANNEL_0].DAR = (uint32_t)&audio_buffer[PingPong+2][0];
        DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_0);
}

/*---------------------------------------------------------------------------------------------------------*/
/* PDMA Callback function                                                                                  */
/*---------------------------------------------------------------------------------------------------------*/
void PDMA1_TA_Callback(uint32_t status)
{
           DrvPDMA_CHSoftwareReset (eDRVPDMA_CHANNEL_1 );
           DrvPDMA_CHEnablelTransfer (eDRVPDMA_CHANNEL_1 );
}
```

PDMA1_Callback runs when a frame transfers end. Use pingpong buffer for next transfer.

```
/*---------------------------------------------------------------------------------------------------------*/
/* I2S/DPWM TX Callback                                                                                    */
/*---------------------------------------------------------------------------------------------------------*/
void PDMA1_Callback(uint32_t status)
{
    static uint8_t PingPong, TxSilence;
    volatile uint32_t u32SFR;

    if(status == I2S_INITIALIZE){
        // Callback can be called from compressor to initialize
        // the buffers.
        PingPong = 0;
        TxSilence =1;
        // Point PDMA to silence buffer
          PDMA->channel[eDRVPDMA_CHANNEL_1].SAR = (uint32_t)&zero_buf;
          PDMA->channel[eDRVPDMA_CHANNEL_1].BCR = ZERO_BUF_SAMPLES *
sizeof(int16_t);

        DrvPDMA_CHEnablelTransfer (eDRVPDMA_CHANNEL_1 );
    }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_1].SAR =
(uint32_t)&audio_buffer[PingPong+2][0];
                    PDMA->channel[eDRVPDMA_CHANNEL_1].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
                        PDMA->channel[eDRVPDMA_CHANNEL_1].SAR = (uint32_t)&zero_buf;
                        PDMA->channel[eDRVPDMA_CHANNEL_1].BCR = ZERO_BUF_SAMPLES *
sizeof(int16_t);
                }else{
                    // Data ready
                        PDMA->channel[eDRVPDMA_CHANNEL_1].SAR =
(uint32_t)&audio_buffer[PingPong+2][0];
                        PDMA->channel[eDRVPDMA_CHANNEL_1].BCR = MAX_FRAMESIZE *
sizeof(int16_t);
                    TxSilence =0;
                }
            }// if(TxSilence) else
            DrvPDMA_CHEnablelTransfer(eDRVPDMA_CHANNEL_1);
        } // if(status == I2S_INITIALIZE) else
}
```

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
V1.00.001	Sep.8, 2011	● Created