SPC2188 SDFM 使用指南

概述

本应用指南旨在为电机控制工程师和嵌入式系统开发者提供全面而实用的指导，以有效利用 Sigma Delta 调制（ΔΣ调制）技术和 SPC2188 等系列微控制器内置的 Sigma Delta 滤波模块（SDFM）。

目录

图片列表

表格列表

版本历史

版本	日期	作者	状态	变更
C/0	2024-03-05	何序清	Released	首次发布

术语或缩写

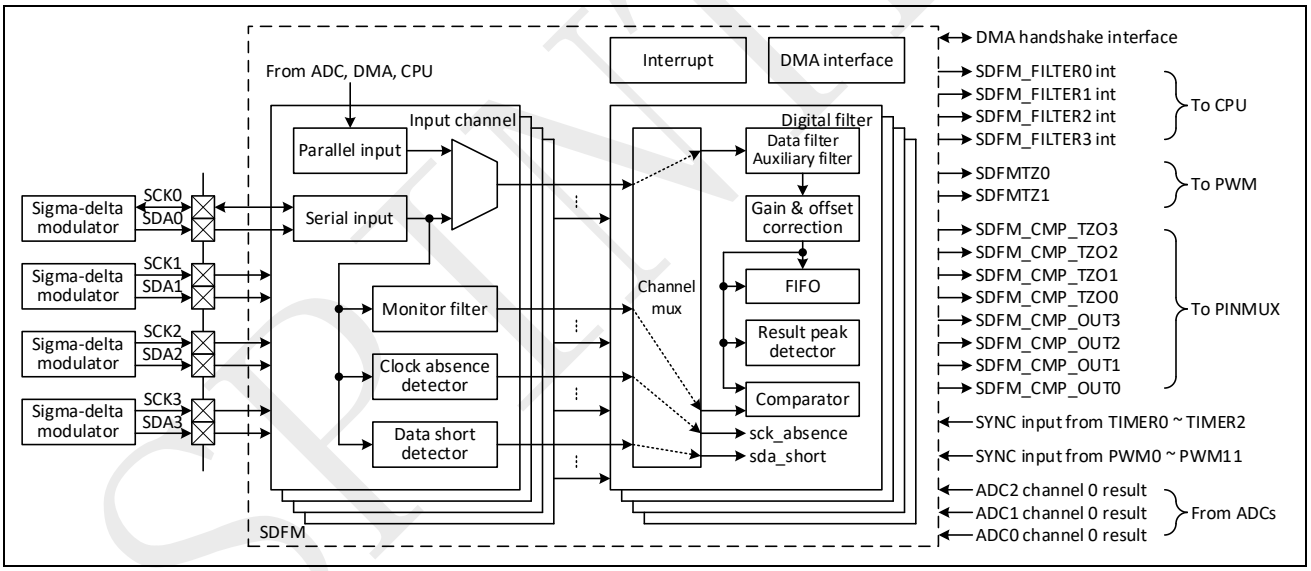
术语或缩写	描述
MCU	Microcontroller Unit，微控制器单元

1 SDFM 特性

SPC2188 的 SDFM 单元可以用作以下功能：

- 4 个外部串行输入通道，支持 SPI 或者 Manchester 编码格式，且极性可配，其中串行时钟频率范围为 1MHz 到 20MHz；
- 支持 1 个串行通道时钟输出；
- 支持来自 ADC 转换结果或者 CPU/DMA 通过总线写入的值作为并行通道输入数据；
- 每个输入通道支持 1 个监视滤波模块，其滤波器类型可配置为：SINCFAST、SINC1、SINC2、SINC3，最大过采样率 128；
- 支持每个外部串行输入通道的采样时钟丢失或数据短路检测；
- 支持 4 个滤波模块（每个滤波模块对应的输入通道可配；每个模块支持的滤波器类型可配置为 SINCFAST、SINC1、SINC2、SINC3、SINC4、SINC5，最大过采样率 1024；支持可选后级辅助滤波器，滤波类型固定为 SINC1 最大过采样率 256；支持数据输出校正、阈值比较；提供 1 个数据输出 FIFO；）；
- 支持产生 PWM 封锁事件；
- 支持 DMA 硬件接口访问滤波结果；

图 1-1：SDFM 框图

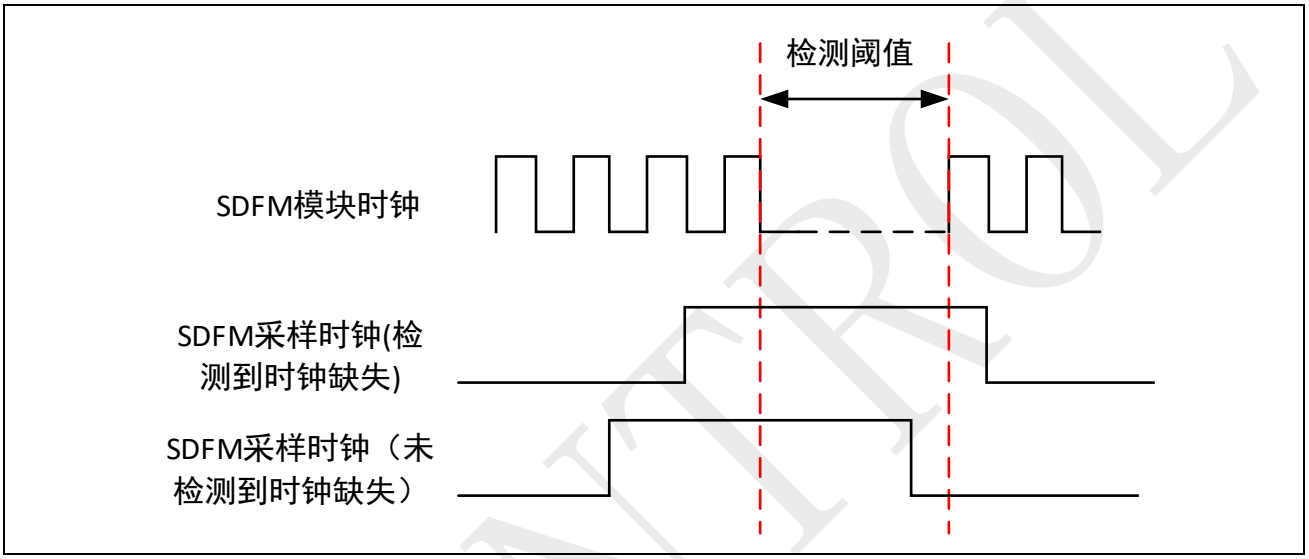


2 SDFM 实例

2.1 时钟缺失检测

本示例演示 SDFM 检测采样时钟缺失。示例使用 CLK_ABSENCE_DETEDT_THRESHOLD 宏进行配置时钟检测的阈值，当在阈值内 SDFM 的采样时钟有变化，则时钟未缺失。如果在阈值内无变化，则时钟缺失如图 2-1 所示。本示例阈值配置为 241 时，不会检测到时钟缺失，当阈值配置为 240 时，会检测到时钟缺失。

图 2-1：采样时钟缺失检测



其程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 SDFM 工作在串行模式；
- 配置 SDFM 的采样时钟源为 SDFM 通道 0，并设置采样时钟的分频系数；
- 配置采样时钟缺失的阈值和使能采样时钟缺失中断；
- 使能 SDFM；

采样时钟缺失检测

```
#include <stdio.h>
#include "spc2188.h"

/* clock absence detect threshold */
#define CLK_ABSENCE_DETEDT_THRESHOLD        241


int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();
```

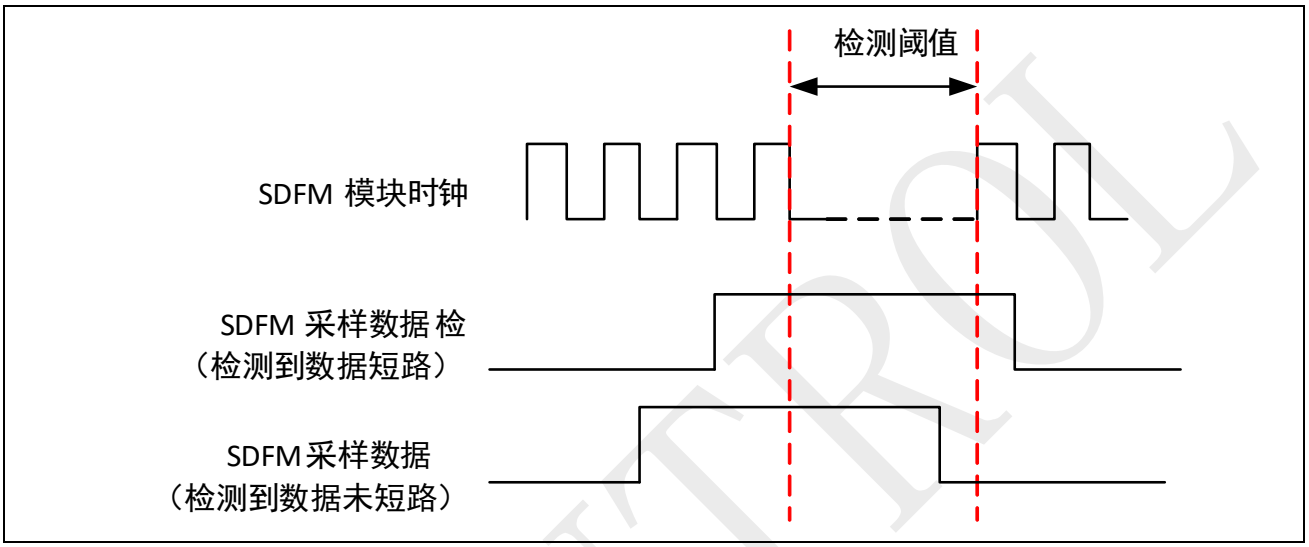
```
    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_SDFM_SCK0);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_SDFM_SDA0);

    /* Init SDFM Serial Mode */
    SDFM_SerialInit(SDFM, SDFM_CH0, SDFM_FLT0);

    /* Set Sample clock comes from channel 0 input SCK0 */
    SDFM_SetSerialSampleClockSource(SDFM, SDFM_CH0,
SDFM_SAMPLE_CLOCK_FROM_CH0_SCK);

    /* Enable FIFO request interrupt */
    SDFM_EnableInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ |
SDFM_INT_SCK_ABSENCE) ;

    /* Enable sample clock output */
    SDFM_EnableSampleClockOutput(SDFM);

    /* Set Sample Clock Div (1~2^16) */
    SDFM_SetSampleClockOutputDiv(SDFM, 480);

    /* Set sample clock absence detect threshold (1 ~ 65536) */
    SDFM_SetSampleClockAbsenceDetectThreshold(SDFM, SDFM_CH0,
CLK_ABSENCE_DETEDT_THRESHOLD);

    /* Enable Sample clock absence detect */
    SDFM_EnableSampleClockAbsenceDetect(SDFM, SDFM_CH0 ) ;

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM */
    SDFM_Enable(SDFM) ;

    /* Enable FILTER0 IRQ */
    NVIC_EnableIRQ(SDFM_FILTER0_IRQn );

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);

    printf("SDFM Test.....\n");

    while(1)
    {
    }
}

void SDFM_FILTER0_IRQHandler(void)
{
    if ( SDFM_GetIntFlag(SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ ))
    {
        SDFM_GetDigitalFilterFIFOData(SDFM, SDFM_FLT0);
        SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ) ;
    }

    if ( SDFM_GetIntFlag(SDFM, SDFM_FLT0, SDFM_INT_SCK_ABSENCE ))
```

```
    {
        printf("SDFM_INT_SCK_ABSENCE\n");
        SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_SCK_ABSENCE ) ;
    }

    SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_GLOBAL ) ;
}
```

2.2 数据短路检测

本示例演示 SDFM 检测数据短路。当在检测阈值内检测到数据没有变化，则数据短路。如果在检测阈值内数据发生变化，则数据未短路如图 2.2 所示。示例使用 ECAP 的 APWM 功能产生一个 1M 的波形当作数据采样，当采样 1M 波形数据时，数据未短路，当采样 GND 时，数据发生短路。

图 2-2：数据短路检测



其程序的配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 APWM 功能使其输出 1M 波形；
- 配置 SDFM 工作在串行模式；
- 配置 SDFM 的采样时钟源为 SDFM 通道 0，并设置采样时钟的分频系数；
- 配置数据短路的阈值和使能数据短路中断；
- 使能 SDFM；

数据短路检测

```
int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_SDFM_SCK0);
```

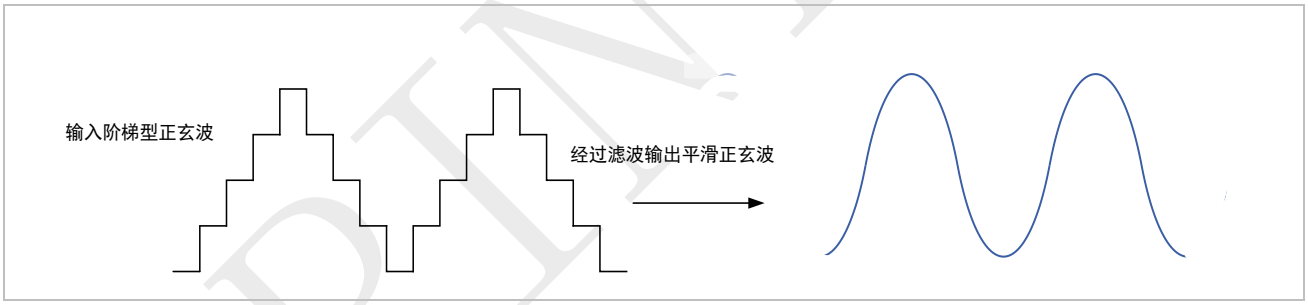
```
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_SDFM_SDA0);

    CLOCK_EnableModule(ECAP_MODULE);

    /* Config ECAP operating mode */
    ECAP_SetMode(ECAP0, ECAP_APWM_MODE);

    /* Calculate Period Value based on system clock and ECAP clock */
    ECAP_SetPRD(ECAP0, CLOCK_GetModuleClock(ECAP_MODULE) / 1000000);

    /* Set polarity */
    ECAP_SetAPWMOutputPolarity(ECAP0, GPIO_LEVEL_HIGH);

    /* Set duty */
    ECAP_APWMSetDuty(ECAP0, 5000);

    /* Set Output */
    PIN_SetChannel(PIN_GPIO64, PIN_GPIO64_ECAP0_APWMO);

    /* Run Counter */
    ECAP_RunCounter(ECAP0);


    /* Init SDFM Serial Mode */
    SDFM_SerialInit(SDFM, SDFM_CH0, SDFM_FLT0);

    /* Set Sample clock comes from channel 0 input SCK0 */
    SDFM_SetSerialSampleClockSource(SDFM, SDFM_CH0,
SDFM_SAMPLE_CLOCK_FROM_CH0_SCK);

    /* Enable FIFO request interrupt */
    SDFM_EnableInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ | SDFM_INT_SDA_SHORT) ;

    /* Enable sample clock output */
    SDFM_EnableSampleClockOutput(SDFM);

    /* Set Sample Clock Div (1~2^16) */
    SDFM_SetSampleClockOutputDiv(SDFM, 480);

    /* Set sample data short detect threshold (1 ~ 256) */
    SDFM_SetShortCircuitDetectThreshold(SDFM, SDFM_CH0, 242);

    /* Enable sample data short detect */
    SDFM_EnableShortCircuitDetect(SDFM, SDFM_CH0);

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM */
    SDFM_Enable(SDFM) ;

    /* Enable FILTER0 IRQ */
    NVIC_EnableIRQ(SDFM_FILTER0_IRQn );

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);

    printf("SDFM Test.....\n");

    while(1)
    {
    }
```

```
}

void SDFM_FILTER0_IRQHandler(void)
{
    if ( SDFM_GetIntFlag(SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ ))
    {
        SDFM_GetDigitalFilterFIFOData(SDFM, SDFM_FLT0);
        SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ) ;
    }

    if ( SDFM_GetIntFlag(SDFM, SDFM_FLT0, SDFM_INT_SDA_SHORT ))
    {
        printf("SDFM_INT_SDA_SHORT\n");
        SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_SDA_SHORT ) ;
    }

    SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_GLOBAL ) ;
}
```

2.3 串行滤波触发中断

本示例演示 SDFM 对阶梯型的正玄波数据进行滤波，输入的信号按照比特串行输入到 SDFM 的数据引脚进行 SDFM 滤波，然后将 SDFM 滤波后的数据经过 MATLAB 进行显示。

图 2-3：串行滤波



其程序的配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 SDFM 工作在串行模式；
- 配置 SDFM 的采样时钟源为 SDFM 通道 0；
- 使能 FIFO 请求中断以及使能通道、数字滤波；
- 使能 SDFM；

串行滤波中断

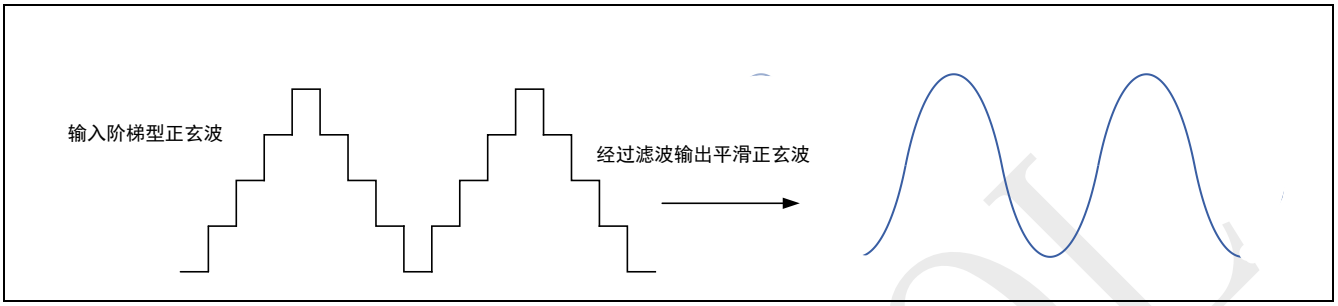
```
int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();
```

```
    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_SDFM_SCK0);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_SDFM_SDA0);

    /* Init SDFM Serial Mode */
    SDFM_SerialInit(SDFM, SDFM_CH0, SDFM_FLT0);

    SDFM_SetSerialSampleClockSource(SDFM, SDFM_CH0,
SDFM_SAMPLE_CLOCK_FROM_CH0_SCK);

    /* Enable FIFO request interrupt */
    SDFM_EnableInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ) ;

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM */
    SDFM_Enable(SDFM) ;

    /* Enable FILTER0 IRQ */
    NVIC_EnableIRQ(SDFM_FILTER0_IRQn );

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);

    printf("SDFM Test.....\n");

    while(1)
    {
    }
}

void SDFM_FILTER0_IRQHandler(void)
{
    if ( SDFM_GetIntFlag(SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ ))
    {
        printf("filter result: %d\n", SDFM_GetDigitalFilterFIFOData(SDFM,
SDFM_FLT0));

        SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_FIFO_REQ) ;
    }

    SDFM_ClearInt( SDFM, SDFM_FLT0, SDFM_INT_GLOBAL );
}
```

2.4 串行滤波触发 DMA 传输

本示例演示 SDFM 对阶梯型的正玄波数据进行滤波，输入的信号按照比特串行输入到 SDFM 的数据引脚进行 SDFM 滤波，然后将 SDFM 滤波后的数据经过 MATLAB 进行显示。

图 2-4：串行滤波



其程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 SDFM 工作在串行模式；
- 使能 SDFM 的 DMA 功能，并且配置 DMA 传输相关信息；
- 使能 FIFO 请求中断以及使能通道、数字滤波；
- 使能 SDFM；

串行滤波 DMA 传输

```
#include <stdio.h>

#include "spc2188.h"

int32_t i32Data = 0;
uint32_t u32DataRxDone;


void SDFM_DMA_Config(void)
{
    /* Enable DMA */
    DMA_Enable(DMAC1);

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Init peripheral (SDFM.FLT0) to memory, pop SDFM.FLT0 FIFO to memory */
    DMA_SetTransferType(&DMAC1->DMACH[0], DMA_PERIPHERAL_TO_MEMORY ) ;
    DMA_SetBlockTransferSize(&DMAC1->DMACH[0], 1) ;
    DMA_DisableSuspend(&DMAC1->DMACH[0]) ;
    DMA_EnableChannelInt(&DMAC1->DMACH[0]) ;

    /* Set source config */
    DMA_SetSourceAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&SDFM-
>SDFMFDAT[SDFM_FLT0] ) ) ;
    DMA_SetSourceAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_NO_CHANGE ) ;
    DMA_SetSourceBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
```

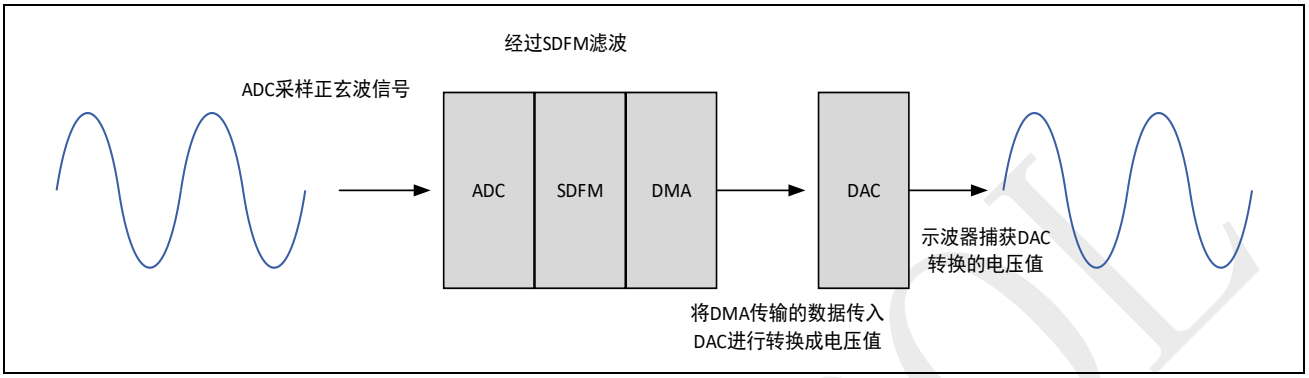
```
    DMA_SetSourceTransferWidth(&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetSourceHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;
    DMA_SetSourcePeripheral(&DMAC1->DMACH[0], DMA_SPER_SDFM_FILTER0 ) ;

    /* Set destinationAddr Config */
    DMA_SetDestinationAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&i32Data) ) ;
    DMA_SetDestinationAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_MODE_INCREASE ) ;
    DMA_SetDestinationBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
    DMA_SetDestinationTransferWidth (&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetDestinationHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_SOFTWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(&DMAC1->DMACH[0]);

    DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
}

int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Config SDFM GPIO */
    PIN_SetChannel(PIN_GPIO16, PIN_GPIO16_SDFM_SCK0);
    PIN_SetChannel(PIN_GPIO17, PIN_GPIO17_SDFM_SDA0);

    /* Init SDFM Serial Mode */
    SDFM_SerialInit(SDFM, SDFM_CH0, SDFM_FLT0);

    /* Enable DMAC1 IRQ */
    NVIC_EnableIRQ( DMAC1_IRQn );

    /* DMA request source */
    SDFM_SetDMARequestSource(SDFM, SDFM_FLT0, SDFM_DMA_REQ_BY_DATA_READY);

    /* DMA enable */
    SDFM_EnableDMA(SDFM, SDFM_FLT0) ;

    /* SDFM DMA Config */
    SDFM_DMA_Config();

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM */
    SDFM_Enable(SDFM) ;

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);
```

```
    printf("SDFM Test.....\n");

    while(1)
    {
        while (u32DataRxDone)
        {
            /* Disable DMA */
            DMA_Disable(DMAC0);
            DMA_SetDestinationAddr(&DMAC1->DMACH[0],
(uint32_t)(long)(&i32Data) ) ;
            /* Enable DMA */
            DMA_Enable(DMAC0);
            u32DataRxDone = 0;
            /* Enable Channel 0 transfer*/
            DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
        }
    }
}

void DMAC1_IRQHandler(void)
{
    /* Get TRANSFER ERROR flag */
    if (DMA_GetGlobalIntFlag(DMAC1, DMA_INT_TRANSFER_ERROR) != 0)
    {
        printf("DMA Channel Error!\n");
    }

    /* Get DMA_CH0 TRANSFER COMPLETE flag */
    if (DMA_GetTransferCompleteIntFlag(DMAC1, DMA_CH0) != 0)
    {
        printf("filter result: %d\n", i32Data);

        /* Clear DMA1 CH0 TRANSFER COMPLETE flag */
        DMA_ClearTransferCompleteInt(DMAC1, DMA_CH0);
        u32DataRxDone = 1;
    }
}
```

2.5 并行滤波触发 DMA 传输

本示例演示 ADC 对正玄波数据进行采样，然后将 ADC 采样后的数据并行的传给 SDFM 进行滤波。当 SDFM 滤波完后通过 DMA 进行传输，最后由 DAC 进行映射滤波后的波形。

图 2-5：滤波过程



其程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 SDFM 工作在并行模式，并设置并行模式的数据源为 ADC；
- 初始化 ADC 功能；
- 使能 SDFM 的 DMA 功能，并且配置 DMA 传输相关信息；
- 配置 DAC 功能；
- 使能 FIFO 请求中断以及使能通道、数字滤波；
- 使能 SDFM；

并行滤波 DMA 传输

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "spc2188.h"


int32_t i32Data = 0;
uint32_t u32DataRxDone;


/*
 * @brief Calculate SDFM Gain
 **/
int64_t SDFM_GetGain(SDFM_SincFilterTypeEnum eType, uint32_t u32OSR, uint32_t
u32AOSR)
{
    int64_t i64Gain = 0;;

    if ( eType == SDFM_FASTSINC )
    {
        i64Gain = (int64_t)pow( u32OSR, 2 ) * 2 * u32AOSR ;
```

```
    }
    else
    {
        i64Gain = (int64_t)pow( u32OSR, eType ) * u32AOSR ;
    }

    return i64Gain;
}


void SDFM_DMA_Config(void)
{
    /* Enable DMA */
    DMA_Enable(DMAC1);

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Init peripheral (SDFM.FLT0) to memory, pop SDFM.FLT0 FIFO to memory */
    DMA_SetTransferType(&DMAC1->DMACH[0], DMA_PERIPHERAL_TO_MEMORY ) ;
    DMA_SetBlockTransferSize(&DMAC1->DMACH[0], 1) ;
    DMA_DisableSuspend(&DMAC1->DMACH[0]) ;
    DMA_EnableChannelInt(&DMAC1->DMACH[0]) ;

    /* Set source config */
    DMA_SetSourceAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&SDFM-
>SDFMFDAT[SDFM_FLT0] ) ) ;
    DMA_SetSourceAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_NO_CHANGE ) ;
    DMA_SetSourceBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
    DMA_SetSourceTransferWidth(&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetSourceHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;
    DMA_SetSourcePeripheral(&DMAC1->DMACH[0], DMA_SPER_SDFM_FILTER0 ) ;

    /* Set destinationAddr Config */
    DMA_SetDestinationAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&i32Data) ) ;
    DMA_SetDestinationAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_MODE_INCREASE ) ;
    DMA_SetDestinationBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
    DMA_SetDestinationTransferWidth (&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetDestinationHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_SOFTWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(&DMAC1->DMACH[0]);

    DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
}


void SDFM_DAC_Config(void)
{
    /* enable DAC0 */
    COMP_EnableDAC(DAC0);

    /* Set DAC0 as Direct mode(DAC code is immediately update) */
    COMP_SetDACCodeLoadTiming(DAC0, DIRECT_LOAD_MODE);

    /* DAC Buffer Initial, only DACBUF0_FROM_DAC0 ~ DACBUF0_FROM_DAC3 connect
to DACBUF0*/
    COMP_SetDACBuffer0Input(DACBUF0_FROM_DAC0);
```

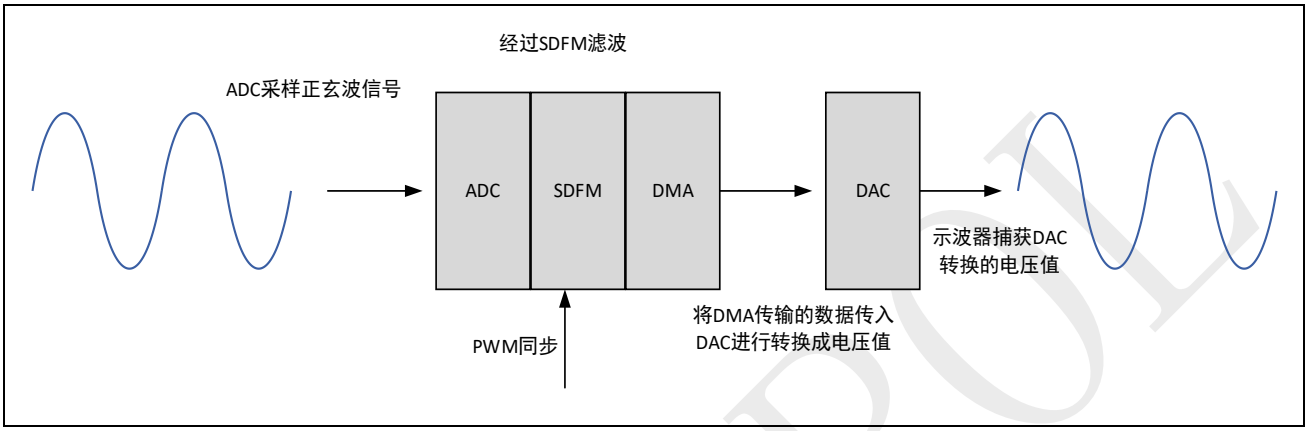
```
    COMP_EnableDACBuffer(DACBUF0);

    /* Enable DAC Buffer Output To GPIO */
    COMP_EnableDACBufferOutputToGPIO(DACBUF0);
}

int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init SDFM Serial Mode */
    SDFM_ParallelInit(SDFM, SDFM_CH0, SDFM_FLT0,
SDFM_PARALLEL_FROM_ADC_RESULT0);

    /*ADC Init*/
    ADC_Init(ADC0, ADC_CH0, ADC0_SH0_P_ANA_IN6, ADC0_SH0_N_GND,
ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Enable DMAC1 IRQ */
    NVIC_EnableIRQ( DMAC1_IRQn );

    /* DMA request source */
    SDFM_SetDMARequestSource(SDFM, SDFM_FLT0, SDFM_DMA_REQ_BY_DATA_READY);

    /* DMA enable */
    SDFM_EnableDMA(SDFM, SDFM_FLT0) ;

    /* SDFM DMA Config */
    SDFM_DMA_Config();

    /* SDFM DAC Config */
    SDFM_DAC_Config();

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM module */
    SDFM_Enable(SDFM) ;

    printf("SDFM Test.....\n");
    ADC_ForceChannelSOC(ADC0, ADC_CH0);

    while(1)
    {

        if (u32DataRxDone)
        {
            DMA_SetDestinationAddr(&DMAC1->DMACH[0],
(uint32_t)(long)(&i32Data) ) ;
            u32DataRxDone = 0;
```

```
            /* Enable Channel 0 transfer*/
            DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
        }

        /* Wait for the ADC sample data conversion to complete */
        while(ADC_GetSHEOCFlag(ADC0, ADC_CH0) == 0) {}

        ADC_ForceChannelSOC(ADC0, ADC_CH0);
    }
}


void DMAC1_IRQHandler(void)
{
    int32_t i32DACCode = 0;

    /* Get TRANSFER ERROR flag */
    if (DMA_GetGlobalIntFlag(DMAC1, DMA_INT_TRANSFER_ERROR) != 0)
    {
        printf("DMA Channel Error!\n");
    }

    /* Get DMA_CH0 TRANSFER COMPLETE flag */
    if (DMA_GetTransferCompleteIntFlag(DMAC1, DMA_CH0) != 0)
    {
        /* Get ADC code */
        i32DACCode = i32Data / SDFM_GetGain(SDFM_SINC2, 16, 1);

        /* Convert ADC code to DAC code */
        i32DACCode = i32DACCode / 8;
        COMP_SetDACCode(DAC0, i32DACCode);

        /* Clear DMA_CH0 TRANSFER COMPLETE flag */
        DMA_ClearTransferCompleteInt(DMAC1, DMA_CH0);
        u32DataRxDone = 1;
    }
}
```

2.6 并行滤波，PWM 进行同步

本示例演示 ADC 对正玄波数据进行采样，然后将 ADC 采样后的数据并行的传给 SDFM 进行滤波，并且 PWM 计数到某个数值时进行同步。当 SDFM 滤波完后通过 DMA 进行传输，最后由 DAC 进行映射滤波后的波形。

图 2-6：滤波过程



其程序配置流程如下：

- 初始化系统时钟，UART 调试口以及初始化 SDFM 的 GPIO；
- 配置 SDFM 工作在并行模式，并设置并行模式的数据源为 ADC；
- 初始化 ADC 功能；
- 使能 SDFM 的 DMA 功能，并且配置 DMA 传输相关信息；
- 配置 DAC 功能；
- 使能 FIFO 请求中断以及使能通道、数字滤波；
- 使能 SDFM；

并行滤波 PWM 同步

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "spc2188.h"

/* This macro is used to get the PWM period with a specified frequency */
#define                 PWMPeriod(u32PWMFreqHz)
((CLOCK_GetModuleClock(PWM_MODULE))/u32PWMFreqHz)/2;
#define                 PWM_FREQ                                10000          /* 20kHz
PWM */
#define                 PWM_DB_NS                               100           /* 1000ns
*/


int32_t i32Data = 0;
uint32_t u32DataRxDone;
```

```
/*
 * @brief Calculate SDFM Gain
 **/
int64_t SDFM_GetGain(SDFM_SincFilterTypeEnum eType, uint32_t u32OSR, uint32_t
u32AOSR)
{
    int64_t i64Gain = 0;;

    if ( eType == SDFM_FASTSINC )
    {
        i64Gain = (int64_t)pow( u32OSR, 2 ) * 2 * u32AOSR ;
    }
    else
    {
        i64Gain = (int64_t)pow( u32OSR, eType ) * u32AOSR ;
    }

    return i64Gain;
}


void SDFM_DMA_Config(void)
{
    /* Enable DMA */
    DMA_Enable(DMAC1);

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Init peripheral (SDFM.FLT0) to memory, pop SDFM.FLT0 FIFO to memory */
    DMA_SetTransferType(&DMAC1->DMACH[0], DMA_PERIPHERAL_TO_MEMORY ) ;
    DMA_SetBlockTransferSize(&DMAC1->DMACH[0], 1) ;
    DMA_DisableSuspend(&DMAC1->DMACH[0]) ;
    DMA_EnableChannelInt(&DMAC1->DMACH[0]) ;

    /* Set source config */
    DMA_SetSourceAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&SDFM-
>SDFMFDAT[SDFM_FLT0] ) ) ;
    DMA_SetSourceAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_NO_CHANGE ) ;
    DMA_SetSourceBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
    DMA_SetSourceTransferWidth(&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetSourceHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_HARDWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;
    DMA_SetSourcePeripheral(&DMAC1->DMACH[0], DMA_SPER_SDFM_FILTER0 ) ;

    /* Set destinationAddr Config */
    DMA_SetDestinationAddr(&DMAC1->DMACH[0], (uint32_t)(long)(&i32Data) ) ;
    DMA_SetDestinationAddrMode(&DMAC1->DMACH[0], DMA_ADDRESS_MODE_INCREASE ) ;
    DMA_SetDestinationBurstLen(&DMAC1->DMACH[0], DMA_BURST_LENGTH_1_ITEM ) ;
    DMA_SetDestinationTransferWidth (&DMAC1->DMACH[0], DMA_TRANSFER_IN_WORD ) ;
    DMA_SetDestinationHandShake(&DMAC1->DMACH[0], DMA_HANDSHAKE_BY_SOFTWARE,
DMA_HANDSHAKE_POL_ACTIVE_HIGH ) ;

    /* Enable the DMA interrupt */
    DMA_EnableTransferCompleteInt(DMAC1, DMA_CH0);

    /* Enable channel 0 interrupt */
    DMA_EnableChannelInt(&DMAC1->DMACH[0]);

    DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
}
```

```
void SDFM_DAC_Config(void)
{
    /* enable DAC0 */
    COMP_EnableDAC(DAC0);

    /* Set DAC0 as Direct mode(DAC code is immediately update) */
    COMP_SetDACCodeLoadTiming(DAC0, DIRECT_LOAD_MODE);

    /* DAC Buffer Initial, only DACBUF0_FROM_DAC0 ~ DACBUF0_FROM_DAC3 connect
to DACBUF0*/
    COMP_SetDACBuffer0Input(DACBUF0_FROM_DAC0);
    COMP_EnableDACBuffer(DACBUF0);

    /* Enable DAC Buffer Output To GPIO */
    COMP_EnableDACBufferOutputToGPIO(DACBUF0);
}



void SDFM_PWM_Config(void)
{
    /* PWM init */
    PWM_InitComplementaryPairChannel(PWM0, PWM_FREQ, PWM_DB_NS);

    /* TBCNT is counting up trigger event */
    PWM_SetSOCCTiming(PWM0, EQU_ZERO);

    /* Enable SOCC signal generation */
    PWM_EnableSOCC(PWM0);

    /* Set the period to generate PWM SOC */
    PWM_SetSOCCPeriod(PWM0, ON_1ST_EVENT);

    /* Set CMPC to PWM0->TBPRD * 2 */
    PWM_SetCMPC(PWM0, PWM0->TBPRD * 2);

    /* Enable PWM counter */
    PWM_RunCounter(PWM0);
}


int main(void)
{
    CLOCK_InitWithRCO(240000000U);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO62, PIN_GPIO62_UART0_TXD);
    PIN_SetChannel(PIN_GPIO63, PIN_GPIO63_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init SDFM Serial Mode */
    SDFM_ParallelInit(SDFM, SDFM_CH0, SDFM_FLT0,
SDFM_PARALLEL_FROM_ADC_RESULT0);

    /* Set digital filter sync event */
    SDFM_SetDigitalFilterSyncSource(SDFM, SDFM_FLT0, SDFM_SYNC_BY_PWM0SOCC);

    PWM_InitComplementaryPairChannel(PWM0, PWM_FREQ, PWM_DB_NS);
```

```
    /* SDFM PWM Config */
    SDFM_PWM_Config();

    /* PWM GPIO Config */
    PIN_SetChannel(PIN_GPIO36, PIN_GPIO36_PWM0A);
    PIN_SetChannel(PIN_GPIO37, PIN_GPIO37_PWM0B);

    /*ADC Init*/
    ADC_Init(ADC0, ADC_CH0, ADC0_SH0_P_ANA_IN6, ADC0_SH0_N_GND,
ADC_SOC_TRIGGER_FROM_SOFTWARE);

    /* Enable DMAC1 IRQ */
    NVIC_EnableIRQ( DMAC1_IRQn );

    /* DMA request source */
    SDFM_SetDMARequestSource(SDFM, SDFM_FLT0, SDFM_DMA_REQ_BY_DATA_READY);

    /* DMA enable */
    SDFM_EnableDMA(SDFM, SDFM_FLT0) ;

    /* SDFM DMA Config */
    SDFM_DMA_Config();

    /* SDFM DAC Config */
    SDFM_DAC_Config();

    /* Force sync Filter0 */
    SDFM_ForceSyncDigitalFilter(SDFM, SDFM_INC_FLT0);

    /* Enable channel and Filter */
    SDFM_EnableChannel(SDFM, SDFM_CH0);
    SDFM_EnableDigitalFilter(SDFM, SDFM_FLT0);

    /* Enable SDFM module */
    SDFM_Enable(SDFM) ;

    printf("SDFM Test.....\n");
    ADC_ForceChannelSOC(ADC0, ADC_CH0);

    while(1)
    {
        if (u32DataRxDone)
        {
            DMA_SetDestinationAddr(&DMAC1->DMACH[0],
(uint32_t)(long)(&i32Data) ) ;
            u32DataRxDone = 0;
            /* Enable Channel 0 transfer*/
            DMA_EnableChannelTransfer(DMAC1, DMA_CH0);
        }

        /* Wait for the ADC sample data conversion to complete */
        while(ADC_GetSHEOCFlag(ADC0, ADC_CH0) == 0) {}

        ADC_ForceChannelSOC(ADC0, ADC_CH0);
    }
}


void DMAC1_IRQHandler(void)
{
    int32_t i32DACCode = 0;

    /* Get TRANSFER ERROR flag */
```

```
    if (DMA_GetGlobalIntFlag(DMAC1, DMA_INT_TRANSFER_ERROR) != 0)
    {
        printf("DMA Channel Error!\n");
    }

    /* Get DMA_CH0 TRANSFER COMPLETE flag */
    if (DMA_GetTransferCompleteIntFlag(DMAC1, DMA_CH0) != 0)
    {
        /* Get ADC code */
        i32DACCode = i32Data / SDFM_GetGain(SDFM_SINC2, 16, 1);

        /* Convert ADC code to DAC code */
        i32DACCode = i32DACCode / 8;
        COMP_SetDACCode(DAC0, i32DACCode);

        /* Clear DMA_CH0 TRANSFER COMPLETE flag */
        DMA_ClearTransferCompleteInt(DMAC1, DMA_CH0);
        u32DataRxDone = 1;
    }
}
```