ECEN 5813

Chutao Wei

Curry Buscher

PES Project 6 Code pdf

Only project6_p2 code is shown, because project6_p2 uses all code from project6_p1

readme.md

```
# cu-ecen-5813-project-6
**Title:**
PES Project 6 Readme <br/>
**Name:**
Curry Buscher, Chutao Wei <br/>
**Repository Comments:** <br/>
In documents folder: <br/>
There is PES Project 6.pdf and PES Project 6 code.pdf<br/>
<br/>
In project6_p1/source folder: <br/>
*project6_p1.c/h*: is the wrapper for freertos system.<br/>
*logger.c/h*: contains debug printing function<br/>
*sin.c/h*: contains code to create sin table in both float or uint16_t
format<br/>
*dac_adc.c/h*: contains code to use DAC, ADC, and summary function<br/>
<br/>
In project6_p2/source folder: <br/>
*project6_p2.c/h*: is the wrapper for freertos system.<br/>
*led.c/h*: contains RGB LED control functions<br/>
*gpio.c/h*: contains gpio control functions<br/>
*logger.c/h*: contains debug printing function<br/>
*dac_adc.c/h*: contains code to use DAC, ADC, DMA and DSP function<br/>
<br/>
(see more details in PES Project 6.pdf) <br/>
<br/>
**Project Comments:**
Please use semihost <br/>

### **Installation/Execution/Editing Notes:**<br/>

**Language:**
C<br/>
**Compiler:**
GCC version 7.4.0<br/>
**IDE:**
MCUExpresso<br/>
**Build Environment:**
Ubuntu 16 or up<br/>
**Target Environment:**
KL25Z<br/>
**License:**
MIT<br/>
```

project6_p2.c (main.c)

```
/*
 * Copyright 2016-2020 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice,
this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice,
this
 *   list of conditions and the following disclaimer in the documentation
and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from
this
 *   software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    project6_p1.c
 * @brief   Application entry point.
 */
/* Standard includes. */
#include <assert.h>
#include <stdio.h>
#include <string.h>
#include <stdio.h>

/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
```

```c
#include "timers.h"
#include "semphr.h"
#include "queue.h"
#include "task.h"

/* Freescale includes. */
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_dma.h"
#include "fsl_dmamux.h"


/* My Own */
#include "sin.h"
#include "dac_adc.h"
#include "led.h"
#include "logger.h"
/*******************************************************************************
***
 * Definitions

*******************************************************************************
*/

/* The software timer period. */
#define SW_TIMER_PERIOD_MS (100 / portTICK_PERIOD_MS)

/*******************************************************************************
***
 * Prototypes

*******************************************************************************
*/
/* The callback function. */
void SwTimerCallback(TimerHandle_t xTimer); // highest priority 5
void DAC_task(void *threadp); // priority 4
void ADC_task(void *threadp); // priority 3
void DMA_task(void *threadp); // priority 2
void DSP_task(void *threadp); // priority 1

/*******************************************************************************
***
 * Global Variables

*******************************************************************************
*/
SemaphoreHandle_t xBinary_DAC;
SemaphoreHandle_t xBinary_ADC;
SemaphoreHandle_t xBinary_DMA;
SemaphoreHandle_t xBinary_DSP;
SemaphoreHandle_t xBinary_LED;
uint32_t Hundredmsec = 0;
```

```
uint32_t target_Hundredmsec;
bool timer_flag;
uint8_t round_count = 0;
/*******************************************************************************
***
 * Code

*******************************************************************************
*/
/*
 * @brief   Application entry point.
 */
/*!
 * @brief Main function
 */
int main(void)
{
    TimerHandle_t SwTimerHandle = NULL;

    /* Init board hardware. */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    SystemCoreClockUpdate();

    init_table();
    init_table_uint16();
    DAC_ADC_DMA_Init();
    init_LED();

    PRINTF("Hello Project 6 Problem 2\n");

    /* Create the Binary */
    xBinary_DAC = xSemaphoreCreateBinary();
    if(xBinary_DAC == NULL)
    {
        PRINTF("Cannot create xBinary_DAC");
    }

    xBinary_ADC = xSemaphoreCreateBinary();
    if(xBinary_ADC == NULL)
    {
        PRINTF("Cannot create xBinary_ADC");
    }

    xBinary_DMA = xSemaphoreCreateBinary();
    if(xBinary_DMA == NULL)
    {
        PRINTF("Cannot create xBinary_DMA");
    }

    xBinary_DSP = xSemaphoreCreateBinary();
    if(xBinary_DSP == NULL)
    {
        PRINTF("Cannot create xBinary_DSP");
    }
```

```
    xBinary_LED = xSemaphoreCreateBinary();
    if(xBinary_LED == NULL)
    {
        PRINTF("Cannot create xBinary_LED");
    }
    xSemaphoreGive(xBinary_LED);

    /* Create the task, storing the handle. */
    BaseType_t xReturned[4];
    TaskHandle_t xHandle[4] = {NULL};
    xReturned[0] = xTaskCreate(DAC_task,
"DAC_task",200,NULL,configMAX_PRIORITIES-2, &xHandle[0]);
    if (xReturned[0] != pdPASS)
    {
        PRINTF("Cannot create DAC_task");
    }
    xReturned[1] = xTaskCreate(ADC_task,
"ADC_task",200,NULL,configMAX_PRIORITIES-3, &xHandle[1]);
    if (xReturned[1] != pdPASS)
    {
        PRINTF("Cannot create ADC_task");
    }
    xReturned[2] = xTaskCreate(DMA_task,
"DMA_task",200,NULL,configMAX_PRIORITIES-4, &xHandle[2]);
    if (xReturned[1] != pdPASS)
    {
        PRINTF("Cannot create DMA_task");
    }

    xReturned[3] = xTaskCreate(DSP_task,
"DSP_task",200,NULL,configMAX_PRIORITIES-5, &xHandle[3]);
    if (xReturned[1] != pdPASS)
    {
        PRINTF("Cannot create DSP_task");
    }

    /* Create the software timer. */
    SwTimerHandle = xTimerCreate("SwTimer",          /* Text name. */
                                 SW_TIMER_PERIOD_MS, /* Timer period. */
                                 pdTRUE,             /* Enable auto reload.
*/
                                 0,                  /* ID is not used. */
                                 SwTimerCallback);    /* The callback function.
*/

    /* Routine explanation */

/*************************************************************************
***
     * 1. Timer task runs every 0.1 sec and release timer Binary
     * 2. DAC task runs when timer Binary release. When done, release DAC
Binary .
     * 3. ADC task runs when DAC Binary release, store result in a buffer
     *
     * After 50 rounds of step 1 to step 3, all tasks are ended. And a DSP
summary
     * will be printed out
```

```
*********************************************************************************
*/

    /* Start timer. */
    xTimerStart(SwTimerHandle, 0);
    /* Start scheduling. */
    vTaskStartScheduler();


    /* Infinite While */
    while(1);// should never be here
}

/*!
 * @brief Software timer callback.xBinary_timer
 */
void SwTimerCallback(TimerHandle_t xTimer)
{
    taskENTER_CRITICAL();
    xSemaphoreGive(xBinary_DAC);
    Hundredmsec++;
    if (timer_flag ==true)
    {
        if(Hundredmsec == target_Hundredmsec)
        {
            timer_flag = false;
            turn_LED_blue(off);
            xSemaphoreGive(xBinary_LED);
        }

    }
    taskEXIT_CRITICAL();
}

unsigned int timerGetRunTimeHundredmsec(void)
{

    return Hundredmsec;
}

/*!
 * @brief DAC task
 */
void DAC_task(void *threadp)
{
    static int i = 0;
    while(1)
    {
        xSemaphoreTake(xBinary_DAC,portMAX_DELAY);
        if(xSemaphoreTake(xBinary_LED,1)==pdTRUE)
        {
            turn_LED_green(on);
            DAC_Write(i);
            turn_LED_green(off);
            xSemaphoreGive(xBinary_LED);
        }
```

```
        else
        {
            DAC_Write(i);
        }

        xSemaphoreGive(xBinary_ADC);
        if(i==50) i = 0;
        else i++;

        if(round_count == 5)vTaskSuspend(NULL);
    }
}

/*!
 * @brief ADC task
 */
void ADC_task(void *threadp)
{
    static int i = 0;
    while(1)
    {
        xSemaphoreTake(xBinary_ADC,portMAX_DELAY);
        ADC_Read(i);

        if(i==64)
        {
            xSemaphoreGive(xBinary_DMA);

            i = 0;
        }
        else i++;

        if(round_count == 5)vTaskSuspend(NULL);
    }
}

/*!
 * @brief ADC task
 */
void DMA_task(void *threadp)
{
    while(1)
    {
        xSemaphoreTake(xBinary_DMA,portMAX_DELAY);
        if(xSemaphoreTake(xBinary_LED,1)==pdTRUE)
        {
            turn_LED_blue(on);
            target_Hundredmsec = Hundredmsec+3;
            timer_flag = true;
        }
        DMA_Transfer();
        xSemaphoreGive(xBinary_DSP);
        round_count++;
        if(round_count == 5)vTaskSuspend(NULL);

    }
}
```

```
void DSP_task(void *threadp)
{
    while(1)
    {
        xSemaphoreTake(xBinary_DSP,portMAX_DELAY);
        DSP_Summary(round_count);
        if(round_count == 5) while(1);
    }
}
```

gpio.c

```
/*
 * gpio.c
 *
 *  Created on: Feb 11, 2020
 *      Author: chutao
 *
 *  Minic the functions from fsl_gpio.c
 *  Still use MKL25Z4.h for hardware addresses
 */

/********************** Include **********************/

#include <stdio.h>
#include <stdint.h>
#include "gpio.h"



/********************** Function **********************/

void set_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
	port->PSOR = (0x1 << pin);
}

void clear_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
	port->PCOR = (0x1 << pin);
}

void toggle_GPIO_Pinout(GPIO_Type *port, uint32_t pin)
{
```

```
	port->PTOR = (0x1 << pin);
}

void init_GPIO_Pin(GPIO_Type *port, uint32_t pin,
		gpio_pin_direct_t pin_direction, uint8_t pin_data)
{
	if (pin_direction == GPIO_DigitalInput)
	{
		// Set pin to input direction
		port->PDDR &= ~(0x1 << pin);
	}
	else if (pin_direction == GPIO_DigitalOutput)
	{
		// Set pin to output direction
		port->PDDR |= (0x1 << pin);
		if (pin_data)
		{
			set_GPIO_Pinout(port,pin);
		}
		else
		{
			clear_GPIO_Pinout(port,pin);
		}
	}
	else
	{
#ifdef LOGGING_DEBUG
		// TODO: Debug message
#endif
	}
}
```

led.c

```
/*
 * led.c
 *
 *  Created on: Feb 11, 2020
 *      Author: chutao
 */



/*************** Include ****************/

#include <logger.h>
#include <stdint.h>
#include "gpio.h"
#include "led.h"
#include "timer.h"

/*************** Global Varibles ****************/

led_color_t color = red;
const char * led_color_string[3] ={"off","on","toggle"};



/*************** Function ****************/

void init_LED(void)
{
	init_GPIO_Pin(LED3_RED_PORT,   LED3_RED_PIN,  GPIO_DigitalOutput, 1);
	init_GPIO_Pin(LED3_GREEN_PORT,	LED3_GREEN_PIN,	GPIO_DigitalOutput, 1);
	init_GPIO_Pin(LED3_BLUE_PORT, LED3_BLUE_PIN,	GPIO_DigitalOutput, 1);
}
```

```
void turn_LED(led_state_t LED_state)
{
	if (color == red)
	{
		turn_LED_red(LED_state);
	}
	else if (color == green)
	{
		turn_LED_green(LED_state);
	}
	else if (color == blue)
	{
		turn_LED_blue(LED_state);
	}
	else
	{
		LOG_ERROR("Unexpected led_state_t");
	}
}

void change_LED_color(led_color_t LED_color)
{
	color = LED_color;
}

void turn_LED_red(led_state_t LED_state)
{
	LOG_DEBUG("Turn LED red %s",led_color_string[LED_state]);
	color = red;
	if (LED_state == off)
```

```
        {
                set_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
        }
        else if (LED_state == on)
        {
                clear_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
        }
        else if (LED_state == toggle)
        {
                toggle_GPIO_Pinout(LED3_RED_PORT, LED3_RED_PIN);
        }
        else
        {
                LOG_ERROR("Unexpected led_state_t");
        }
}

void turn_LED_green(led_state_t LED_state)
{
        LOG_DEBUG("Turn LED green %s",led_color_string[LED_state]);
        color = green;
        if (LED_state == off)
        {
                set_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
        }
        else if (LED_state == on)
        {
                clear_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
        }
        else if (LED_state == toggle)
        {
```

```
            toggle_GPIO_Pinout(LED3_GREEN_PORT, LED3_GREEN_PIN);
        }
        else
        {
            LOG_ERROR("Unexpected led_state_t");
        }
}

void turn_LED_blue(led_state_t LED_state)
{
        LOG_DEBUG("Turn LED blue %s",led_color_string[LED_state]);
        color = blue;
        if (LED_state == off)
        {
            set_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
        }
        else if (LED_state == on)
        {
            clear_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
        }
        else if (LED_state == toggle)
        {
            toggle_GPIO_Pinout(LED3_BLUE_PORT, LED3_BLUE_PIN);
        }
        else
        {
            LOG_ERROR("Unexpected led_state_t");
        }
}
```

logger.c

```
/*
 * log.c
 *
 *  Created on: Dec 18, 2018
 *      Author: Chutao Wei
 *
 */
/********************** Include **********************/

#include <stdint.h>
#include "logger.h"
#include "project6_p2.h"
/********************** Global **********************/

log_status_t log_status = disable;


/********************** Functions **********************/

/**
 * @return a timestamp value for the logger, typically based on a free
running timer.
 * This will be printed at the beginning of each log message.
 */

unsigned int loggerGetTimestamp(void)
{
    return (unsigned int)(timerGetRunTimeHundredmsec());
}
#ifdef HMS_FORMAT
uint32_t loggerGetTimestampHour()
{
    uint64_t msec = timerGetRunTimeMilliseconds();
    return (uint32_t)(msec/360000);
}
uint32_t loggerGetTimestampMinute()
{
    uint64_t msec = timerGetRunTimeMilliseconds();
    return (uint32_t)(msec/6000);
}
uint32_t loggerGetTimestampSecond()
{
    uint64_t msec = timerGetRunTimeMilliseconds();
    return (uint32_t)(msec/1000);
}
uint32_t loggerGetTimestampTenthSec()
{
    uint64_t msec = timerGetRunTimeMilliseconds();
    return (uint32_t)(msec/100);
}
#endif
```

dac_adc.c

```
/*
 * dac_adc.c
 *
 *  Created on: Apr 27, 2020
 *      Author: chutao
 */
/* Kernel includes. */
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "semphr.h"
#include "queue.h"
#include "task.h"

#include "dac_adc.h"
#include "fsl_dac.h"
#include "fsl_adc16.h"
#include "fsl_debug_console.h"
#include "fsl_dma.h"
#include "fsl_dmamux.h"
#include "MKL25Z4.h"
#include "sin.h"
#include "project6_p2.h"
#include "math.h"
#include "logger.h"
/*******************************************************************************
***
 * Defines

*******************************************************************************
*/
#define DEMO_ADC16_CHANNEL_GROUP 0U
#define DEMO_ADC16_USER_CHANNEL 0U /* PTE20, ADC0_SE0 */
#define NUM_POINTS 50
#define DMA_CHANNEL 0
#define DMA_SOURCE 63
#define BUF_SIZE 64

/*******************************************************************************
***
 * Global Variables

*******************************************************************************
*/
volatile bool g_Adc16ConversionDoneFlag = false;
volatile uint32_t g_Adc16ConversionValue = 0;
adc16_channel_config_t g_adc16ChannelConfigStruct;

dma_handle_t g_DMA_Handle;
volatile bool g_Transfer_Done = false;

uint16_t ADC_buf[BUF_SIZE];
uint16_t DSP_buf[BUF_SIZE];

unsigned int DMA_start_time;
```

```
unsigned int DMA_end_time;
/*******************************************************************************
***
 * Code

******************************************************************************
*/
// code provided by the fsl library
uint16_t average(uint16_t arr[], uint32_t n)
{
    int i;
    int average = arr[0];
    for (i = 0; i < n; i++)
        average = average + arr[i];
    return (average/n);
}
// source: https://www.geeksforgeeks.org/c-program-find-largest-element-
array/
uint16_t max(uint16_t arr[], uint32_t n)
{
    int i;
    int max = arr[0];
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

uint16_t min(uint16_t arr[], uint32_t n)
{
    int i;
    int min = arr[0];
    for (i = 1; i < n; i++)
        if (arr[i] < min)
            min = arr[i];
    return min;
}
// from https://www.programiz.com/c-programming/examples/standard-deviation
uint16_t SD(uint16_t arr[], uint16_t mean,  uint32_t n)
{
    uint32_t variance = 0;
    int i;
    for (i = 0; i < 10; ++i)
        variance += pow(arr[i] - mean, 2);
    return sqrt(variance / n);
}

// interrupt hanlder
void ADC0_IRQHandler(void)
{
    g_Adc16ConversionDoneFlag = true;
    /* Read conversion result to clear the conversion completed flag. */
    g_Adc16ConversionValue = ADC16_GetChannelConversionValue(ADC0,
DEMO_ADC16_CHANNEL_GROUP);
}

/* User callback function for DMA transfer. */
```

```
void DMA_Callback(dma_handle_t *handle, void *param)
{
    g_Transfer_Done = true;
}
// init function
void DAC_ADC_DMA_Init(void)
{
    EnableIRQ(ADC0_IRQn);
    adc16_config_t adc16ConfigStruct;
    dac_config_t dacConfigStruct;

    /* Configure the DAC. */
    /*
     * dacConfigStruct.referenceVoltageSource =
kDAC_ReferenceVoltageSourceVref2;
     * dacConfigStruct.enableLowPowerMode = false;
     */
    DAC_GetDefaultConfig(&dacConfigStruct);
    DAC_Init(DAC0, &dacConfigStruct);
    DAC_Enable(DAC0, true); /* Enable output. */

    /* Configure the ADC16. */
    /*
     * adc16ConfigStruct.referenceVoltageSource =
kADC16_ReferenceVoltageSourceVref;
     * adc16ConfigStruct.clockSource = kADC16_ClockSourceAsynchronousClock;
     * adc16ConfigStruct.enableAsynchronousClock = true;
     * adc16ConfigStruct.clockDivider = kADC16_ClockDivider8;
     * adc16ConfigStruct.resolution = kADC16_ResolutionSE12Bit;
     * adc16ConfigStruct.longSampleMode = kADC16_LongSampleDisabled;
     * adc16ConfigStruct.enableHighSpeed = false;
     * adc16ConfigStruct.enableLowPower = false;
     * adc16ConfigStruct.enableContinuousConversion = false;
     */
    ADC16_GetDefaultConfig(&adc16ConfigStruct);
#if defined(BOARD_ADC_USE_ALT_VREF)
    adc16ConfigStruct.referenceVoltageSource =
kADC16_ReferenceVoltageSourceValt;
#endif
    ADC16_Init(ADC0, &adc16ConfigStruct);

    /* Make sure the software trigger is used. */
    ADC16_EnableHardwareTrigger(ADC0, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) &&
FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_Success == ADC16_DoAutoCalibration(ADC0))
    {
        //PRINTF("\r\nADC16_DoAutoCalibration() Done.");
    }
    else
    {
        //PRINTF("ADC16_DoAutoCalibration() Failed.\r\n");
    }
#endif /* FSL_FEATURE_ADC16_HAS_CALIBRATION */

    /* Prepare ADC channel setting */
    g_adc16ChannelConfigStruct.channelNumber = DEMO_ADC16_USER_CHANNEL;
```

```
    g_adc16ChannelConfigStruct.enableInterruptOnConversionCompleted = true;

#if defined(FSL_FEATURE_ADC16_HAS_DIFF_MODE) &&
FSL_FEATURE_ADC16_HAS_DIFF_MODE
    g_adc16ChannelConfigStruct.enableDifferentialConversion = false;
#endif /* FSL_FEATURE_ADC16_HAS_DIFF_MODE */
    /* Configure DMAMUX */
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, DMA_CHANNEL, DMA_SOURCE);
    DMAMUX_EnableChannel(DMAMUX0, DMA_CHANNEL);
    /* Configure DMA one shot transfer */
    DMA_Init(DMA0);
    DMA_CreateHandle(&g_DMA_Handle, DMA0, DMA_CHANNEL);
    DMA_SetCallback(&g_DMA_Handle, DMA_Callback, NULL);
}

void DAC_Write(int i)
{
    LOG_DEBUG("DAC_Write\n");
    DAC_SetBufferValue(DAC0, 0U, sin_lookup_table_uint16[i]);
}

void ADC_Read(int i)
{
    LOG_DEBUG("ADC_Read\n");
    g_Adc16ConversionDoneFlag = false;
    ADC16_SetChannelConfig(ADC0, DEMO_ADC16_CHANNEL_GROUP,
&g_adc16ChannelConfigStruct);

    while (!g_Adc16ConversionDoneFlag);
    ADC_buf[i] = g_Adc16ConversionValue;
}

void DMA_Transfer(void)
{
    LOG_DEBUG("DMA_Transfer\n");
    DMA_start_time = timerGetRunTimeHundredmsec();
    dma_transfer_config_t transferConfig;
    DMA_PrepareTransfer(&transferConfig, ADC_buf, sizeof(ADC_buf[0]),
            DSP_buf, sizeof(DSP_buf[0]), sizeof(ADC_buf),
kDMA_MemoryToMemory);
    DMA_SubmitTransfer(&g_DMA_Handle, &transferConfig, kDMA_EnableInterrupt);
    DMA_StartTransfer(&g_DMA_Handle);
    while (g_Transfer_Done != true);
    DMA_end_time = timerGetRunTimeHundredmsec();
}

void DSP_Summary(uint8_t round_count)
{
    PRINTF("DSP Summary Report %d\t \n",round_count);


    uint16_t DSP_ave = average(DSP_buf, BUF_SIZE);
    uint16_t DSP_max = max(DSP_buf, BUF_SIZE);
    uint16_t DSP_min = min(DSP_buf, BUF_SIZE);
    uint16_t standard_deviation = SD(DSP_buf,DSP_ave,BUF_SIZE);
```

```
    float DSP_ave_f = 3.3/4096*DSP_ave;
    float DSP_max_f = 3.3/4096*DSP_max;
    float DSP_min_f = 3.3/4096*DSP_min;
    float standard_deviation_f = 3.3/4096*standard_deviation;
    PRINTF("Ave\t\tMax\t\tMin\t\tSD\t\t \n");
    PRINTF("%f\t%f\t%f\t%f\t
\n",DSP_ave_f,DSP_max_f,DSP_min_f,standard_deviation_f);

    PRINTF("\t \n");
}
```