```
/******************************************************************************
 *                                PES PROJECT 6
 *             AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                  Cross Platform IDE: MCUXpresso IDE v11
 *                         Cross-Compiler: ARM GCC
 *                         freertos_tickless.c
 * REFERENCES:-
 * https://www.programiz.com/c-programming/examples/standard-deviation
 *                         MCU SDK FREE RTOS EXAMPLES                             *
 ******************************************************************************/

/**************Header files**************/
#include "FreeRTOS.h"
#include "task.h"
#include "timers.h"
#include "semphr.h"
#include "queue.h"
#include "logger.h"
#include "dacadc.h"
#include "queue_adc.h"
#include "dma.h"
#include "cir_buffer.h"
#include "led.h"
#include "fsl_device_registers.h"
#include "fsl_debug_console.h"
#include "fsl_port.h"
#include "fsl_gpio.h"
#include "board.h"
#include "fsl_port.h"
#include "pin_mux.h"
#include "fsl_common.h"
#include "clock_config.h"
#include <math.h>
#include <stdint.h>
#include <stdio.h>
//#include "fsl_dac.h"
//#include "fsl_adc16.h"
#if configUSE_TICKLESS_IDLE
#include "fsl_lptmr.h"
#endif

//#define ON 0
//#define OFF 1
////#define QUE_MODE == ON
//#define QUE_MODE == OFF
//static QueueHandle_t adc_queue = NULL;
/*******************************************************************************
 * Definitions
 ******************************************************************************/
extern unsigned long long timecount;
//static QueueHandle_t adc_queue = NULL;
//static QueueHandle_t dsp_queue = NULL;
//static SemaphoreHandle_t xSemaphore_led;

//static SemaphoreHandle_t xSemaphore_producer;
```

```
adc16_channel_config_t g_adc16ChannelConfigStruct;
volatile uint16_t g_Adc16ConversionValue = 0;

#define VREF_BRD 3.300
#define SE_12BIT 4096.0

#define BOARD_SW_GPIO BOARD_SW2_GPIO
#define BOARD_SW_PORT BOARD_SW2_PORT
#define BOARD_SW_GPIO_PIN BOARD_SW2_GPIO_PIN
#define BOARD_SW_IRQ BOARD_SW2_IRQ
#define BOARD_SW_IRQ_HANDLER BOARD_SW2_IRQ_HANDLER
#define BOARD_SW_NAME BOARD_SW2_NAME

/* @brief FreeRTOS tickless timer configuration. */
#define BOARD_LPTMR_IRQ_HANDLER LPTMR0_IRQHandler /*!< Timer IRQ handler. */
#define TICKLESS_LPTMR_BASE_PTR LPTMR0            /*!< Tickless timer base address.
*/
#define TICKLESS_LPTMR_IRQn LPTMR0_IRQn           /*!< Tickless timer IRQ number. */

/* Task priorities. */
/* clang-format off */
#define dac_task_PRIORITY   ( configMAX_PRIORITIES - 4 )
#define adc_task_PRIORITY   ( configMAX_PRIORITIES - 4 )
#define dsp_math_PRIORITY ( configMAX_PRIORITIES - 4 )

//#define store_buffer1_PRIORITY ( configMAX_PRIORITIES - 3 )
//#define transfer_buffer2_PRIORITY ( configMAX_PRIORITIES - 3 )

#define TIME_DELAY_SLEEP      100

/* Interrupt priorities. */
#define SW_NVIC_PRIO 2

volatile uint16_t var_sinefunc;
volatile uint8_t n=0,t3flag=0;
volatile uint8_t num_execute=1;
volatile bool g_Adc16ConversionDoneFlag = false;
//uint8_t ucHeap[ configTOTAL_HEAP_SIZE ];

TaskHandle_t TaskHandle_1;

ring_buffer *adc_buff;
uint16_t dsp_buff[100];
ring_status receive_status;
/**************of no use ******************/
 static const unsigned int sine_values[] =
        {

                          2482U, 2638U, 2791U, 2939U, 3080U, 3212U, 3332U, 3438U,
3530U, 3605U, 3663U, 3701U, 3721U, 3721U, 3702U, 3663U,
                          3606U, 3531U, 3440U, 3333U, 3213U, 3082U, 2941U, 2793U,
2640U, 2484U, 2329U, 2176U, 2028U, 1886U, 1755U, 1635U,
                          1528U, 1436U, 1360U, 1303U, 1264U, 1244U, 1243U, 1263U,
1301U, 1358U, 1433U, 1524U, 1630U, 1750U, 1881U, 2022U,
                          2170U, 2323U, 2478U
        };


/* clang-format on */
/*******************************************************************************
 * Prototypes
 ******************************************************************************/
extern void vPortLptmrIsr(void);
LPTMR_Type *vPortGetLptrmBase(void);
IRQn_Type vPortGetLptmrIrqn(void);

/*******************************************************************************
 * Variables
 ******************************************************************************/
void sine_func(void);
static void DAC_converter(void *pvParameters);
static void ADC_converter(void *pvParameters);
static void SW_task(void *pvParameters);
static void dsp_math(void *pvParameters);
SemaphoreHandle_t xSWSemaphore = NULL;
/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Main function
 */


void DEMO_ADC16_IRQ_HANDLER_FUNC(void)
{
    g_Adc16ConversionDoneFlag = true;
    /* Read conversion result to clear the conversion completed flag. */
    g_Adc16ConversionValue = ADC16_GetChannelConversionValue(DEMO_ADC16_BASEADDR,
DEMO_ADC16_CHANNEL_GROUP);

}


int main(void)
{
/* Define the init structure for the input switch pin */
#ifdef BOARD_SW_NAME
    gpio_pin_config_t sw_config = {
        kGPIO_DigitalInput, 0,
    };
#endif
#if configUSE_TICKLESS_IDLE
    lptmr_config_t lptmrConfig;

    CLOCK_EnableClock(kCLOCK_Lptmr0);
    /* Configuration LPTMR  */
    /*
```

```
     * lptmrConfig->timerMode = kLPTMR_TimerModeTimeCounter;
     * lptmrConfig->pinSelect = kLPTMR_PinSelectInput_0;
     * lptmrConfig->pinPolarity = kLPTMR_PinPolarityActiveHigh;
     * lptmrConfig->enableFreeRunning = false;
     * lptmrConfig->bypassPrescaler = true;
     * lptmrConfig->prescalerClockSource = kLPTMR_PrescalerClock_1;
     * lptmrConfig->value = kLPTMR_Prescale_Glitch_0;
     */
    LPTMR_GetDefaultConfig(&lptmrConfig);
    LPTMR_Init(TICKLESS_LPTMR_BASE_PTR, &lptmrConfig);
    /* Enable timer interrupt */
    LPTMR_EnableInterrupts(TICKLESS_LPTMR_BASE_PTR, kLPTMR_TimerInterruptEnable);
    /* Enable at the NVIC */
    EnableIRQ(TICKLESS_LPTMR_IRQn);
#endif
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    EnableIRQ(DEMO_ADC16_IRQn);
    DAC_ADC_Init();
//    xSemaphore_led = xSemaphoreCreateMutex();

//#if QUE_MODE == ON
//    adc_queue = xQueueCreate(64, 32);
// //   dsp_queue = xQueueCreate(64, 32);
//#endif

    /* Print a note to terminal. */

   // PRINTF("Tickless Demo example\r\n");
#ifdef BOARD_SW_NAME
    PRINTF("Press %s to wake up the CPU\r\n", BOARD_SW_NAME);
    /* Init input switch GPIO. */
    PORT_SetPinInterruptConfig(BOARD_SW_PORT, BOARD_SW_GPIO_PIN,
kPORT_InterruptFallingEdge);
    NVIC_SetPriority(BOARD_SW_IRQ, SW_NVIC_PRIO);
    EnableIRQ(BOARD_SW_IRQ);
    GPIO_PinInit(BOARD_SW_GPIO, BOARD_SW_GPIO_PIN, &sw_config);
#endif

    /*Create tickless task*/

    xTaskCreate(DAC_converter, "DAC_converter", configMINIMAL_STACK_SIZE + 90, NULL,
dac_task_PRIORITY, NULL);
    xTaskCreate(ADC_converter, "ADC_converter", configMINIMAL_STACK_SIZE + 90, NULL,
adc_task_PRIORITY, NULL);

   // xTaskCreate(dsp_math,"dsp_math", configMINIMAL_STACK_SIZE + 38, NULL,
dsp_math_PRIORITY,TaskHandle_1);
  //  xTaskCreate(STORE_BUFFER_1, "STORE_BUFFER_1", configMINIMAL_STACK_SIZE + 38,
NULL, store_buffer1_PRIORITY, NULL);
  //  xTaskCreate(TRANS_BUFFER_2, "TRANS_BUFFER_2", configMINIMAL_STACK_SIZE + 38,
NULL, transfer_buffer2_PRIORITY, NULL);
   // xTaskCreate(SW_task, "SW_task", configMINIMAL_STACK_SIZE + 90, NULL,
dac_task_PRIORITY, NULL);
```

```
    /*Task Scheduler*/
    PRINTF("/*****************************************TASKS
BEGIN************************************/");
    vTaskStartScheduler();
    for (;;)
        ;
}

/* Tickless Task */
/******************DAC_converter**********************************
This task gives the mathematical values that generate the sine wave with
sine wave that runs from 1V to 3V. Period from peak to peak should
 be 5 seconds with a .1 second step.
************************************************************************/

static void DAC_converter(void *pvParameters)
{
       log_messages(mode,DACconverter);
       uint16_t val_sinefunc;
        if(mode == debug)
        {
    PRINTF("\r\nTick count : ");
        }
    for (;;)
    {
        if(mode == debug)
        {
        PRINTF("%d \n\r", xTaskGetTickCount());
        }
        //PRINTF("\r\n my sine count :\r\n");

        /******************Semaphore mutex
implementation**********************************
        mutex based semaphore is used to share LED between the tasks.
        *************************************************************************/


//        if( xSemaphore_led != NULL )
//        {             /* See if the mutex can be obtained.  If the mutex is not
available             wait 10 ticks to see if it becomes free. */
//           if( xSemaphoreTake( xSemaphore_led, 0 ) == pdTRUE )
//           {
             init_LED();
             green_led();
       //Green LED is toggled
//            xSemaphoreGive( xSemaphore_led );
//           }
//        }
    // sine_func();

       var_sinefunc = ((2 + sin((2*3.14*n)/(float)50)) / 3.3)*4096;
        if(mode == debug)
        {
             PRINTF("\n \r SIN FUNCTION VALUES ARE %d",var_sinefunc);
        }
        DAC_SetBufferValue(DEMO_DAC_BASEADDR, 0U, var_sinefunc);
        vTaskDelay(TIME_DELAY_SLEEP);
//        j++;
//        if (j == 51)
//        {
//        j = 0;
//        }

      n++;
                   if( n == 51 )
                   {
                          n = 0;
                   }

    }
}

/*****************ADC_converter*********************************
This task is used to store the ADC values in circular buffer/queues .
Since we have a buffer of capacity 64,after 64 values it will
initiate DMA transfer
*********************************************************************/

static void ADC_converter(void *pvParameters)
{
      log_messages(mode,ADCconverter);

      uint8_t count=0;
      // float voltRead;
      uint16_t voltRead;


//#if QUE_MODE == OFF
      adc_buff = (ring_buffer*)malloc(sizeof(ring_buffer));
       receive_status = buff_initialize(adc_buff, 64);
//#endif

       if(mode == debug)
              {
       PRINTF("RECEIVE STATUS for buff initialization is %d",receive_status);
              }

       for (;;)
       {
              g_Adc16ConversionDoneFlag = false;
              ADC16_SetChannelConfig(DEMO_ADC16_BASEADDR, DEMO_ADC16_CHANNEL_GROUP,
&g_adc16ChannelConfigStruct);
              g_Adc16ConversionDoneFlag = true;
              /* Read conversion result to clear the conversion completed flag. */
              g_Adc16ConversionValue =
ADC16_GetChannelConversionValue(DEMO_ADC16_BASEADDR, DEMO_ADC16_CHANNEL_GROUP);
              while (!g_Adc16ConversionDoneFlag)
              {
              }
              PRINTF("\r\n\r\nADC Value: %d\r\n", g_Adc16ConversionValue);

              /* Convert ADC value to a voltage based on 3.3V VREFH on board */
              voltRead = (uint16_t)(g_Adc16ConversionValue * (VREF_BRD / SE_12BIT));
             // PRINTF("\r\nADC Voltage: %0.3f\r\n", voltRead);
              PRINTF("\r\n ADC Voltage: %d\r\n", voltRead);
             // xQueueSend(adc_queue, &voltRead, 10);

                    if(count==64)
                    {

                    PRINTF("\n \r DMA transfer done from ADC TO DSP");
                   PRINTF("\n \r DMA Start time is:");
                    timestamps(timecount);
                    DMA();
                    //#if QUE_MODE == OFF
                    transfer_DMA(adc_buff->buffer,dsp_buff,64*2);
                    //#endif

                    // #if QUE_MODE == ON
                    // transfer_DMA_2(adc_queue,dsp_buff,64*2); //for queue transfer
to array
                    // #endif
                    // #if QUE_MODE == OFF

                    /*****************Semaphore mutex
implementation*********************************
                           mutex based semaphore is used to share LED between the
tasks.

**********************************************************************/
//                  if( xSemaphore_led != NULL )
//                         {             /* See if the mutex can be obtained.  If
the mutex is not available             wait 10 ticks to see if it becomes free. */
//                              if( xSemaphoreTake( xSemaphore_led, 0 ) == pdTRUE )
//                              {
                    uint8_t m;
                                init_LED();
                                BLUE_LED();
                                for(m=0;m<250;m++);
//                               xSemaphoreGive( xSemaphore_led );
//                              }
//                         }
                   // PRINTF("ADC_BUFFER %p",adc_buff->buffer);
                    PRINTF("\n \r DMA End time is:");
                    timestamps(timecount);
                    uint8_t i;
//                                            for(i=0;i<64;i++)
//                                            {
//                                                  PRINTF("\n \r DSP buffer values
are %d",dsp_buff[i]);
//                                            }
                    count=0;
                    // xTaskCreate(dsp_math,"dsp_math", configMINIMAL_STACK_SIZE +
38, NULL, 0,TaskHandle_1);

//                   xQueueReset(adc_queue);
                     xTaskCreate(dsp_math, "dsp_math", configMINIMAL_STACK_SIZE + 38,
NULL, dsp_math_PRIORITY, NULL);
       //             xTaskResumeFromISR( TaskHandle_1 );
                     }
                     else
                     {
                           if(mode==debug){
                     PRINTF("\n\r COUNT LESS THAN 64: %d",count);
                           }
                     count++;
//#if QUE_MODE == OFF
                     receive_status = buff_add_item(adc_buff,g_Adc16ConversionValue);
//#endif
                    // PRINTF("RECEIVE STATUS for add itemmmm of adcccc bufferrrrr
ISSS %d",receive_status);
                     // #if QUE_MODE == ON
                     // printf("\n \r QUEUE mode on");
                     // timestamps(timecount);
                     // uint16_t receive;
                     // xQueueSend(adc_queue, &g_Adc16ConversionValue, 10);
                     // //xQueueReceive( adc_queue, &receive, portMAX_DELAY );
                     // //PRINTF("queue has value %d",receive);
                     // #endif

                     }
              vTaskDelay(TIME_DELAY_SLEEP);
        }
}


/* Switch Task */
static void SW_task(void *pvParameters)
{
    xSWSemaphore = xSemaphoreCreateBinary();
    for (;;)
    {
      PRINTF("IN SW TASK LOOP");
        if (xSemaphoreTake(xSWSemaphore, portMAX_DELAY) == pdTRUE)
        {
            PRINTF("CPU woke up by EXT interrupt\r\n");
        }

    }
}
```

```
/******************DSP_MATH**********************************
This task is used to calculate minimum,maximum,standard deviation and average
of all the values in the dsp buffer.
After 4 cycles,the tasks are terminated.
************************************************************************/
static void dsp_math(void *pvParameters)
{

    log_messages(mode,dspmath);
for(;;)
    {
        float max=0,min=4096,avg=0,sd=0,cal=0,buff_val[64];
        cal = 3.3/4096; //step
        uint8_t count;
        for( count=0;count<64;count++)
        {
            if((*dsp_buff+count) < min)
            {
            min = (*dsp_buff+count);
            }
            else if((*dsp_buff+count)>max)
            {
            max= (*dsp_buff+count);
            }
            avg = avg + (*dsp_buff+count);
//adc_buffer++;
        }
            min = min*cal;
            max = max*cal;
            avg = avg*cal;
            avg = avg/64.00;
        for (uint8_t count2 = 0; count2 < 64;count2++)
        {
            buff_val[count2] = (*dsp_buff+count)*cal;
            sd = sd + pow(((*dsp_buff+count) - avg), 2);
        }
            sd=sqrt(sd/64.00);

        PRINTF("\n \r Min=%f,Max=%f,Avg=%f,sd=%f",min,max,avg,sd);

        PRINTF("\n \r /********************NUM COUNT IS: %d
***********************/",num_execute);
        num_execute++;

            if(num_execute >4)
            {
                PRINTF("\n \r
*****************************************TASKS
SUSPENDED**************************/");
```

```
                          PRINTF("\n \r
*************************************************************************************
************************************************************");
                          //vTaskSuspendAll(TaskHandle_1);
                          vTaskEndScheduler();
                    }

                    vTaskSuspend(TaskHandle_1);

      }
 //num_execute++;
}






/*!
 * @brief Interrupt service fuction of switch.
 *
 * This function to wake up CPU
 */
#ifdef BOARD_SW_NAME
void BOARD_SW_IRQ_HANDLER(void)
{
    portBASE_TYPE xHigherPriorityTaskWoken = pdFALSE;

    /* Clear external interrupt flag. */
    GPIO_ClearPinsInterruptFlags(BOARD_SW_GPIO, 1U << BOARD_SW_GPIO_PIN);

    xSemaphoreGiveFromISR(xSWSemaphore, &xHigherPriorityTaskWoken);
}
#endif
/*!
 * @brief Interrupt service fuction of LPT timer.
 *
 * This function to call vPortLptmrIsr
 */
#if configUSE_TICKLESS_IDLE == 1
void BOARD_LPTMR_IRQ_HANDLER(void)
{
    vPortLptmrIsr();
}

/*!
 * @brief Fuction of LPT timer.
 *
 * This function to return LPT timer base address
 */

LPTMR_Type *vPortGetLptrmBase(void)
{
    return TICKLESS_LPTMR_BASE_PTR;
```

```
}

/*!
 * @brief Fuction of LPT timer.
 *
 * This function to return LPT timer interrupt number
 */

IRQn_Type vPortGetLptmrIrqn(void)
{
    return TICKLESS_LPTMR_IRQn;
}
#endif /* configUSE_TICKLESS_IDLE */
```

```
/******************************************************************************
 *                                   PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                        Cross Platform IDE: MCUXpresso IDE v11
 *                                Cross-Compiler: ARM GCC
 *                                      cir_buffer.c
 ******************************************************************************/
/*********************************REFERENCE**********************************
 https://stackoverflow.com/questions/827691/how-do-you-implement-a-circular-buffer-in-c
 http://www.equestionanswers.com/c/c-circular-buffer.php
*****************************************************************************/
#include "cir_buffer.h"

ring_status r_status;
uint8_t count_add_item = 1;
/**************************BUFFER INITIALIZE FUNCTION**********************
 In this function we take pointer and capacity of the buffer in the function and
  check for errors if pointer is null or capacity is 0 and if not then
  we allocate memory for buffer and initialize all the variables of the structure.
*****************************************************************************/

ring_status buff_initialize(ring_buffer *p, uint8_t capacity)
{
        log_messages(mode,buffinitialize);

        if(p == NULL || capacity <= 0)
        {
                return buffer_init_not_done;
        }
        else
        {
                p->buffer = (uint16_t*) malloc(sizeof(uint8_t)*capacity);
                PRINTF("P BUFFERRRRRRRRRRRRRRR %p",p->buffer);
                p->head = 0;
                p->tail = 0;
                p->capacity = capacity;
                p->count = 0;
                p->head_count=0;
                p->tail_count=0;
                //PRINTF("\n\r Head position in initial stage is at %d",p->head);
                return buffer_init_done;
                if(p == NULL)
                {
                        return buffer_init_not_done;
                }

        }
}
/***************************BUFFER CHECK FULL FUNCTION**********************
 In this function we take pointer in the function and
  check for errors such as if pointer is null then buffer pointer is valid,
  if count=capacity,then returns the status of buffer_full and
  if count=0 then we say buffer is empty else it returns status
  as buffer not full
*****************************************************************************/
ring_status buff_check_full(ring_buffer *p)
{
        log_messages(mode,buffcheck_full);
        if(p == NULL)
        {
                buff_ptr_valid(p);
        }
        else if(p->count == p->capacity)
        {
                return buffer_full;
        }
        else if (p->count == 0)
        {
                return buffer_empty;
        }

                return buffer_not_full;

}
/******************BUFFER CHECK EMPTY FUNCTION**********************
```

```
  In this function we take pointer in the function and
  check for errors such as if pointer is null then buffer pointer is valid,
  if count=capacity,then returns the status of buffer_full and
  if count=0 then we say buffer is empty else it returns status
  as buffer not full
****************************************************************************************/
ring_status buff_check_empty(ring_buffer *p)
{
	log_messages(mode,buffcheck_empty);
	if(p == NULL)
		{
		buff_ptr_valid(p);
		}
	else if(p->count == 0)
		{
			return buffer_empty;
		}
	else if(p->count == p->capacity)
		{
			return buffer_full;
		}

			return buffer_not_empty;
}

/*****************BUFFER ADD ITEM
FUNCTION*******************************************************************
  In this function we take pointer and the character to be stored
  in the function and check for conditions before storing the character such as:
  1-if head =capacity-1 and if buffer is empty or buffer is not full then it wraps around the buffer
  2-if buffer is full,then returns the status of buffer_full and item is not added to buffer
  3-if buffer is empty or not full then it increments the head for next round of addition
**************************************************************************************************
**************/
ring_status buff_add_item(ring_buffer *p,uint16_t item)
{
	log_messages(mode,buffadd_item);
			if((p->head == (p->capacity - 1)) && ((buff_check_full(p) == buffer_empty) ||
(buff_check_full(p) == buffer_not_full)))
			{

				*(p->buffer + p->head) = item;
//				PRINTF("\n\r Wrap around p->buffer[p->head]: %c",*(p->buffer + p-
>head));
//				PRINTF("\n\r Wrap around address of p->head  %p",(p->buffer + p-
>head));
			//	if(mode==test || mode==debug){
				PRINTF("\n Wrap around occurs");
```

```
                //}
                p->head = 0;
                p->tail = 0;
            //      p->count++;
                p->count = 0;
                p->head_count++;
                return wrap_around;
            }
//          else if(buff_check_full(p) == buffer_full)
//          {
//              p->head = 0;
//
//      //      init_LED();
//      //      error_led();
//          //  if(mode==test || mode==debug)
//              //{
//              PRINTF("\n Item not added to buffer full");
//              //}
//
//              return item_not_added_in_buff;
//          }

            else if((buff_check_full(p) == buffer_empty) || (buff_check_full(p) ==
buffer_not_full))
            {
                *(p->buffer + p->head) = item;
                PRINTF("\n\rTHE ITEM STORED ISSSSSSSS %d",item);

    //PRINTF("***************************************************************");
                PRINTF("\n\r p->buffer[p->head] character: %c",*(p->buffer + p-
>head));

                PRINTF("\n\r address of p->head  %p",(p->buffer + p->head));
                p->head++;
                p->count++;
                p->head_count++;
                //PRINTF("\n\r head position incremented is now %d",p->head);
                //PRINTF("\n\r head count is %d",p->count);
                //PRINTF("\n\r4TH");
                PRINTF("COUNT OF ADD ITEM IS %d",count_add_item);
                count_add_item++;
                return item_added_in_buff;
            }

return 0;
}
/**************************************BUFFER REMOVE ITEM
FUNCTION**************************************
```

```
 In this function we take pointer in the function and check for conditions before storing the character
such as:
 1-if head =capacity-1 and if buffer is empty or buffer is not full then it wraps around the buffer
 2-if buffer is full,then returns the status of buffer_full and item is not added to buffer
 3-if buffer is empty or not full then it increments the tail count for next round of addition
*************************************************************************************
*****************/

ring_status buff_remove_item(ring_buffer *p)
{
    log_messages(mode,buffremove_item);
if(p->head_count > p->tail_count)
{
    uint8_t read;

        if((p->tail == (p->capacity - 1)) && ((buff_check_empty(p) == buffer_full) ||
(buff_check_empty(p) == buffer_not_empty)))
            {
                    read = *(p->buffer + p->tail);
                    //PRINTF("\n\r Wrap around p->buffer[p->tail]: %c",read);
                    //PRINTF("\n\r Wrap around address of p->tail  %p",(p->buffer +
p->tail));
            //      if(mode==test || mode==debug)
                //  {
                        PRINTF("\n Wrap around occurs");
                    //}
                    p->tail = 0;
                    p->tail_count++;
                    p->count--;
                    return wrap_around;
            }

        else if(buff_check_empty(p) == buffer_empty)
        {
            PRINTF("ITEM not removed because full");
            return oldest_item_not_removed;
        }
        else if((buff_check_empty(p) == buffer_full) || (buff_check_empty(p) ==
buffer_not_empty))
            {
                read = *(p->buffer + p->tail);
//              PRINTF("\n\r p->buffer[p->tail] character: %c",read);
//              PRINTF("\n\r address of p->tail  %p",(p->buffer + p->tail));
        //      if(mode==test || mode==debug)
        //      {
                    PRINTF("\n Buffer removed \n");
        //      }
                (p->tail)++;
```

```
                                        p->count--;
                                        p->tail_count++;
                                //      PRINTF("\n\rtail position incremented is now %d",p->tail);
                                //      PRINTF("\n\rcount  now is %d",p->count);
                                        return oldest_item_removed;
                                }
}
else
{
        PRINTF("\n********* wait************ \n");
        //PRINTF("waittttttttttttt");
}
return 0;
}

/**************************************BUFFER
DESTROY*************************************
  In this function we take pointer in the function and destroy the created buffer:
  1-Check if the pointer is null and free buffer
**************************************************************************************
*****************/



ring_status buff_destroy(ring_buffer *p)
{
        log_messages(mode,buffdestroy);
                if(p == NULL)
                {
                buff_ptr_valid(p);
                }
                //else
                //{
                free(p->buffer);
                //free(buffer);
                return buffer_destroyed;
                //}

}

/**************************************BUFFER REMOVE ITEM
FUNCTION*************************************
  In this function we take pointer in the function and check if pointer is valid:
  1-if pointer is null, return status is FAIL
  2-else return status is SUCCESS
**************************************************************************************
*****************/
```

```
ring_status buff_ptr_valid(ring_buffer *p)
{
	log_messages(mode,buffptr_valid);
	if(p == NULL)
		{
			return FAIL;
		}
	else
	{
		return SUCCESS;
	}
}


/**************************************BUFFER REMOVE ITEM
FUNCTION**************************************
  In this function we take pointer in the function and reallocate more memory on the heap
  1-takes input of the previous malloc pointer
  2-allocates more memory in the same address or at new address on the heap; if memory is reallocated
  then it return status as memory reallocated
  3-if new memory is not reallocated then it returns a status as memory not reallocated
**************************************************************************************
*****************/


ring_status buff_resize(ring_buffer *p)
{
	log_messages(mode,buffresize);
	if(p->count == p->capacity)
	{
		//putstr("\n \r *******************************************************");
		PRINTF("\n \r Buffer is resized");

		p->buffer_new = (uint16_t*) realloc(p->buffer, sizeof(char)*2*(p->capacity));

		return memory_reallocated;
	}

	else
	{
		return memory_not_reallocated;
	}
}

/*******************************************************************************
 *                                  PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                           Cross Platform IDE: MCUXpresso IDE v11
 *                                                     Cross-Compiler: ARM GCC
 *                                                               cir_buffer.h
 ******************************************************************************/
/*********************************REFERENCE*******************************
 https://stackoverflow.com/questions/827691/how-do-you-implement-a-circular-buffer-in-c
****************************************************************************/

/*
 * Header File CIR_BUFFER_H_
 * Contains:-      Includes needed for cir_buffer.c file,
 *                                     ring buffer structure,
 *                                     ring status structure,
 *                                     functions in the cir_buffer.c file
 */

#ifndef CIR_BUFFER_H_
#define CIR_BUFFER_H_

//#include "uartpoll.h"
//#include "uartinterrupt.h"
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include "board.h"
//#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "logger.h"
//#include "led.h"

typedef struct ring_buf{
   uint16_t *buffer;
   uint8_t head;
   uint8_t tail;
   uint8_t capacity;
   uint8_t count;
   uint8_t head_count;
   uint8_t tail_count;
   uint16_t *buffer_new;
}ring_buffer;
```

```
typedef enum{
SUCCESS = 0,
FAIL,
buffer_full,
buffer_not_full,
buffer_empty,
buffer_not_empty,
buffer_init_done,
buffer_init_not_done,
buffer_ptr_valid,
buffer_ptr_invalid,
buffer_destroyed,
buffer_not_destoryed,
item_added_in_buff,
item_not_added_in_buff,
oldest_item_removed,
oldest_item_not_removed,
wrap_around,
memory_reallocated,
memory_not_reallocated
}ring_status;

ring_status buff_initialize(ring_buffer *p, uint8_t capacity);
ring_status buff_check_full(ring_buffer *p);
ring_status buff_check_empty(ring_buffer *p);
ring_status buff_add_item(ring_buffer *p,uint16_t item);
ring_status buff_remove_item(ring_buffer *p);
ring_status buff_ptr_valid(ring_buffer *p);
ring_status buff_resize(ring_buffer *p);
ring_status buff_destroy(ring_buffer *p);
#endif




/****************************************************************************
 *                              PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                      Cross Platform IDE: MCUXpresso IDE v11
 *                              Cross-Compiler: ARM GCC
 *                                    dacadc.c
 ******************************************************************************/
#include "dacadc.h"

extern adc16_channel_config_t g_adc16ChannelConfigStruct;
//volatile uint32_t g_Adc16ConversionValue = 0;

void DAC_ADC_Init(void)
```

```c
{
    adc16_config_t adc16ConfigStruct;
    dac_config_t dacConfigStruct;

    /* Configure the DAC. */
    /*
     * dacConfigStruct.referenceVoltageSource = kDAC_ReferenceVoltageSourceVref2;
     * dacConfigStruct.enableLowPowerMode = false;
     */
    DAC_GetDefaultConfig(&dacConfigStruct);
    DAC_Init(DEMO_DAC_BASEADDR, &dacConfigStruct);
    DAC_Enable(DEMO_DAC_BASEADDR, 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(DEMO_ADC16_BASEADDR, &adc16ConfigStruct);

    /* Make sure the software trigger is used. */
    ADC16_EnableHardwareTrigger(DEMO_ADC16_BASEADDR, false);
#if defined(FSL_FEATURE_ADC16_HAS_CALIBRATION) && FSL_FEATURE_ADC16_HAS_CALIBRATION
    if (kStatus_Success == ADC16_DoAutoCalibration(DEMO_ADC16_BASEADDR))
    {
        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 */
}
/*****************************************************************************
 *                                  PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                          Cross Platform IDE: MCUXpresso IDE v11
 *                                  Cross-Compiler: ARM GCC
 *                                          dacadc.h
 *****************************************************************************/

#ifndef DACADC_H_
#define DACADC_H_

#include "fsl_debug_console.h"
#include "board.h"
#include "fsl_adc16.h"
#include "fsl_dac.h"
#include <stdint.h>
#include "clock_config.h"
#include "pin_mux.h"
#include "fsl_gpio.h"
/*******************************************************************************
 * Definitions
 ******************************************************************************/
#define DEMO_ADC16_BASEADDR ADC0
#define DEMO_ADC16_CHANNEL_GROUP 0U
#define DEMO_ADC16_USER_CHANNEL 0U /* PTE20, ADC0_SE0 */
#define DEMO_DAC_BASEADDR DAC0

#define DEMO_ADC16_IRQn ADC0_IRQn
#define DEMO_ADC16_IRQ_HANDLER_FUNC ADC0_IRQHandler
#define DAC_1_0_VOLTS 1241U
#define DAC_1_5_VOLTS 1862U
#define DAC_2_0_VOLTS 2482U
#define DAC_2_5_VOLTS 3103U
#define DAC_3_0_VOLTS 3724U




/*******************************************************************************
 * Variables
 ******************************************************************************/


//adc16_channel_config_t g_adc16ChannelConfigStruct;
```

```
/*******************************************************************************
 * Prototypes
 ******************************************************************************/
/* Initialize ADC16 & DAC */
void DAC_ADC_Init(void);



#endif


/*******************************************************************************
 *                                   PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                           Cross Platform IDE: MCUXpresso IDE v11
 *                                  Cross-Compiler: ARM GCC
 *                                         dma.c
 *********************************************************************************/
#include "dma.h"

//#include "cir_buffer.h"
//#include "logger.h"
dma_handle_t g_DMA_Handle;


//ring_status receive_status,receive_status2;
/*******************************************************************************
 * Definitions
 ******************************************************************************/

#define BUFF_LENGTH 4
#define DMA_CHANNEL 0
#define DMA_SOURCE 63
/*******************************************************************************
 * Prototypes
 ******************************************************************************/

/*******************************************************************************
 * Variables
 ******************************************************************************/

volatile bool g_Transfer_Done = false;
/*******************************************************************************
 * Code
 ******************************************************************************/

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

```
void DMA_Callback(dma_handle_t *handle, void *param)
{
   g_Transfer_Done = true;
}

/* Main function. Execute DMA transfer with transactional APIs. */
//int main(void)
//{
//    uint32_t srcAddr[BUFF_LENGTH] = {0x01, 0x02, 0x03, 0x04};
//    uint32_t destAddr[BUFF_LENGTH] = {0x00, 0x00, 0x00, 0x00};
//    uint32_t i = 0;
    dma_transfer_config_t transferConfig;

//    BOARD_InitPins();
//    BOARD_BootClockRUN();
//    BOARD_InitDebugConsole();

//    s_buff = (ring_buffer*)malloc(sizeof(ring_buffer));
//    PRINTF("\n\r %d",sizeof(ring_buffer));
//    receive_status = buff_initialize(s_buff, 10);
//    for(int i=0;i<15;i++){
//    char a='5';
//    buff_add_item(s_buff,a);
//    a++;
//    }

//    d_buff = (ring_buffer*)malloc(sizeof(ring_buffer));
//    PRINTF("\n\r %d",sizeof(ring_buffer));
//    receive_status = buff_initialize(d_buff, 10);
//    for(int i=0;i<15;i++){
//    char b='4';
//    buff_add_item(d_buff,b);
//    b++;
//   }
    /* Print source buffer */
//    PRINTF("\n \r DMA memory to memory transfer example begin.\r\n\r\n");
//    PRINTF("Destination Buffer:\r\n");
//    for (i = 0; i < BUFF_LENGTH; i++)
//    {
//        PRINTF("%d\t", destAddr[i]);
//    }
    /* Configure DMAMUX */
    void DMA(void)
    {
            log_messages(mode,init_dma);
    DMAMUX_Init(DMAMUX0);
    DMAMUX_SetSource(DMAMUX0, DMA_CHANNEL, DMA_SOURCE);
    DMAMUX_EnableChannel(DMAMUX0, DMA_CHANNEL);
    DMA_Init(DMA0);
    }
    /* Configure DMA one shot transfer */
    void transfer_DMA(uint16_t *src,uint16_t *dest,uint8_t transfer_data)
    {
            log_messages(mode,dmatransfer);
    DMA_CreateHandle(&g_DMA_Handle, DMA0, DMA_CHANNEL);
    DMA_SetCallback(&g_DMA_Handle, DMA_Callback, NULL);
    DMA_PrepareTransfer(&transferConfig,src,2,dest,2,transfer_data,kDMA_MemoryToMemory);
    DMA_SubmitTransfer(&g_DMA_Handle, &transferConfig, kDMA_EnableInterrupt);
    DMA_StartTransfer(&g_DMA_Handle);
    /* Wait for DMA transfer finish */
//   }
    while (g_Transfer_Done != true)
    {
    }
    /* Print destination buffer */
//   PRINTF("\r\n\r\nDMA memory to memory transfer example finish.\r\n\r\n");
//   PRINTF("Destination Buffer:\r\n");
//   for(int i=0;i<16;i++)
//   {
//   buff_remove_item(d_buff);
//   d_buff++;
//   }
    }


/*******************************************************************************
 *                                             PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                  Cross Platform IDE: MCUXpresso IDE v11
 *                                             Cross-Compiler: ARM GCC
 *                                                      dma.h
 ******************************************************************************/
#ifndef DMA_H_
#define DMA_H_


#include "cir_buffer.h"
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_dma.h"
#include "fsl_dmamux.h"
#include <stdio.h>
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
```

```
void DMA_Callback(dma_handle_t *handle, void *param);
void DMA(void);
void transfer_DMA(uint16_t *src,uint16_t *dest,uint8_t transfer_data);

#endif /* DMA_H_ */


/*****************************************************************************
 *                                       PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                               Cross Platform IDE: MCUXpresso IDE v11
 *                                       Cross-Compiler: ARM GCC
 *                                               led.h
 *****************************************************************************/

#include "led.h"

void init_LED(void);
void start(void);
void error(void);
void _end(void);

void init_LED(void)
{
   LED_BLUE_INIT(1);
   LED_RED_INIT(1);
   LED_GREEN_INIT(1);
}

void BLUE_LED(void)              //wait to receive data
{
        LED_RED_OFF();
        LED_GREEN_OFF();
        LED_BLUE_TOGGLE();
        delay(10);
}

void error_led(void)     //error or disconnected state
{
        LED_GREEN_OFF();
        LED_BLUE_OFF();
        LED_RED_ON();
        delay(10);
}
```

```
void green_led(void)     //when transmitting data
{
        LED_RED_OFF();
        LED_BLUE_OFF();
        //LED_GREEN_ON();
        LED_GREEN_TOGGLE();
        delay(10);
}


void delay(uint32_t d)
{
        uint32_t count = d*7000;          /****** As clock is 8MHz *****/
        while(count!=0)
        {
count--;/***** Wasting MCU cycles to get the desired delay ******/
        }
}


/****************************************************************************
 *                                   PES PROJECT 5
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                            Cross Platform IDE: MCUXpresso IDE v11
 *                                   Cross-Compiler: ARM GCC
 *                                          led.h
 ****************************************************************************/

#ifndef LED_H_

#define LED_H_
#include <stdint.h>
#include <stdio.h>
#include "board.h"
//#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "logger.h"
#include "fsl_debug_console.h"
#include "cir_buffer.h"



void init_LED(void);
void BLUE_LED(void);
void green_led(void);
void delay(uint32_t d);
void error_led();
#endif


/***************************************************************************
 *                                          PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                  Cross Platform IDE: MCUXpresso IDE v11
 *                                          Cross-Compiler: ARM GCC
 *                                              logger.c
 ****************************************************************************/
/********************************REFERENCE********************************
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf

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

#include "logger.h"

unsigned long long timecount=0;

/*****************SYSTICK HANDLER FOR TIMER*********************************
 WE USE THE SYSTICK HANDLER FOR 10 HZ I.E 0.1 SEC TIME WHICH INTIIATES AT THE START OF THE
PROGRAM
**********************************************************************************
****/
//void Init_Systick()
//{
//SysTick->LOAD =  48000000/100;
//NVIC_SetPriority(SysTick_IRQn,3);
//SysTick->VAL = 0;
//SysTick->CTRL = 0x7;
//}
///*******WE USE THE START AND END CRITICAL SECTIONS HERE**********************/
//void SysTick_Handler(){
//
//timecount++;
//
//}

uint8_t sec=0,min=0,hour=0;


/*****************TIMESTAMPS*********************************
 USING THE TIMECOUNTER FROM SYSTICK HANDLER WE MAKE THE TIMESTAMPS FUNCTION
 WHICH CALCULATES THE HOURS,MINS AND SECS
**********************************************************************************
****/
void timestamps(unsigned long long timer){
```

```
if(timer>1000)
{
timer=0;
}
if(timer==0){
            sec++;
}
if(sec!=0 && sec%60==0)
{
min++;
sec=0;
}
if(min!=0 && min%60==0)
{
hour++;
min=0;
}
if(hour>24)
  {
hour=0;
}

PRINTF("\t %02d:%02d:%02d:%u \n ",hour,min,sec,timer);
}


/*****************LOG LEVEL FUNCTIONS*********************************
                                                  PRINTS THE TYPE OF MODE USED
*********************************************************************/
void log_level(log_mode mode)
{
   if(mode == debug)
   {
       PRINTF("\n\rMODE: Debug");
   }
   else if(mode == normal)
   {
      PRINTF("\n\r MODE: Normal");
   }
}

/*****************LOG STRING MESSAGES*********************************
 THIS ARRAY GIVES THE STRING FOR PARTICULAR APPLICATION
*********************************************************************************
****/
char ch_arr[40][40]={   "\t Initialize the buffer",
                                                                          "\t Checks if Buffer is full",
                        "\t Checks if Buffer is Empty",
                        "\t Add element to the Buffer",
                        "\t Remove element from the Buffer",
                        "\t Checks if Pointer to Buffer is valid",
                        "\t Destroys the Buffer",
                        "\t Resizes the Buffer",
                        "\t TASK 1=DAC ADC Function"
                        "\t TASK 2=ADC Converter"
                        "\t TASK 3=DSP math function"
                        "\t init_dma"
                        "\t dmatransfer"

                    };

void logger_func(log_func func_nm)
{
    if(func_nm == buffinitialize)
    {
        //PRINTF("\r \t buff_initialize");
        PRINTF("\t buff_initialize");
        PRINTF(ch_arr[0]);
    }
    else if(func_nm == buffcheck_full)
    {
        PRINTF("\t buff_check_full");
        PRINTF(ch_arr[1]);
    }
    else if(func_nm == buffcheck_empty)
    {
        PRINTF("\tbuff_check_empty");
        PRINTF(ch_arr[2]);
    }
    else if(func_nm == buffadd_item)
    {
        PRINTF("\t buff_add_item");
        PRINTF(ch_arr[3]);
    }
    else if(func_nm == buffremove_item)
    {
        PRINTF("\t buff_remove_item");
        PRINTF(ch_arr[4]);
    }
    else if(func_nm == buffptr_valid)
    {
        PRINTF("\t buff_ptr_valid");
        PRINTF(ch_arr[5]);

    }
    else if(func_nm == buffdestroy)
    {
        PRINTF("\t buff_destroy");
        PRINTF(ch_arr[6]);
    }
    else if(func_nm == buffresize)
    {
        PRINTF("\tbuff_resize");
        PRINTF(ch_arr[7]);
    }
    else if(func_nm == DACconverter)
        {
            PRINTF("\t DACconverter");
            PRINTF(ch_arr[8]);
        }
    else if(func_nm ==  ADCconverter)
        {
            PRINTF("\t ADCconverter");
            PRINTF(ch_arr[9]);
        }
    else if(func_nm == dspmath)
        {
            PRINTF("\t dspmath");
            PRINTF(ch_arr[10]);
        }
    else if(func_nm == init_dma)
            {
                PRINTF("\t DMA");
                PRINTF(ch_arr[11]);
            }
    else if(func_nm == dmatransfer)
            {
                PRINTF("\t transfer_dma");
                PRINTF(ch_arr[12]);
            }

}
/*****************LOG
MESSAGES********************************************************
 In this function we print log mode,function name and string along with timestamps
**********************************************************************************
****/
void log_messages(log_mode mode,log_func func_nm)
{
    log_level(mode);
    logger_func(func_nm);
    timestamps(timecount);
}
```

```
/*********************************************************************************
 *                                              PES PROJECT 6
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                    Cross Platform IDE: MCUXpresso IDE v11
 *                                              Cross-Compiler: ARM GCC
 *                                                         logger.h
 *********************************************************************************/

#ifndef LOGGER_H_
#define LOGGER_H_

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>
#include "cir_buffer.h"
#include "board.h"
//#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"


typedef enum
{
	debug,
	normal
}log_mode;

typedef enum
{
	buffinitialize = 0,
	buffcheck_full,
	buffcheck_empty,
	buffadd_item,
	buffremove_item,
	buffptr_valid,
	buffdestroy,
	buffresize,
	DACconverter,
	ADCconverter,
	dspmath,
	init_dma,
	dmatransfer
}log_func;


//#define mode debug //      /**************CHANGE HERE*******************/
#define mode normal

void log_level(log_mode mode);
void logger_func(log_func func_nm);
void log_messages(log_mode mode,log_func func_nm);
//void putstr(unsigned char *string);
#endif
```