```
/*****************************************************************************
*                                    PES PROJECT 5
*          AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
*                                Cross Platform IDE: MCUXpresso IDE v11
*                                Cross-Compiler: ARM GCC
*                                    Project_5.c
*****************************************************************************/

#include "uartinterrupt.h"
#include "uartpoll.h"
#include "cir_buffer.h"
#include "logger.h"
#include "application.h"
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "system.h"
#include "unittest.h"

char str[100];
ring_buffer *t_buff;
ring_buffer *r_buff;
ring_status receive_status;

unsigned char string[100];
int main()
{

        /* Init board hardware. */
  BOARD_InitBootPins();
  BOARD_InitBootClocks();
  BOARD_InitBootPeripherals();
        /* Init FSL debug console. */
  BOARD_InitDebugConsole();

  Init_Systick();
  Init_UART0();
  init_LED();
  wait_receive_led();
/***********************BUFFER FUNCTIONS***********************************/
  r_buff = (ring_buffer*)malloc(sizeof(ring_buffer));
  PRINTF("\n\r %d",sizeof(ring_buffer));
  receive_status = buff_initialize(r_buff, 10);
  PRINTF("\n\r Rx1 status is: %d",receive_status);
```

```
//   receive_status=buff_check_empty(r_buff);
//   PRINTF("\n \r RX status is %d",receive_status);
        if(mode == test)
        {
                putstr("************************TEST MODE ON-START UNIT TEST************************************");
                unit_test();
        }


//while(1){

        /*********APPLICATION AND ECHO MODE*******************************/
#if MODE == APPLICATION_MODE
        log_messages(mode,applicationmode);
        #if UART_MODE == POLLING_MODE

                uart_getstr_poll(str);
                 putstr(str);
                 putstr("\n\n");
                app_mode(str);
#endif

                #if UART_MODE == INTERRUPT_MODE

                #endif

#endif

#if MODE == ECHO_MODE
         log_messages(mode,echomode);
        #if UART_MODE == POLLING_MODE
                char a = UART0_poll_getchar();
                 UART0_poll_putchar(a);

        #elif UART_MODE == INTERRUPT_MODE

        #endif



#endif

//}
return 0;
}
```

```
/*****************************************************************************
 *                                  PES PROJECT 5
 *                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                          Cross Platform IDE: MCUXpresso IDE v11
 *                            Cross-Compiler: ARM GCC
 *                                  uartpoll.c
 *****************************************************************************/
/************************REFERENCE*********************************
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf
*****************************************************************************/

#include "uartpoll.h"
#include "logger.h"
#include "uartinterrupt.h"
#include "cir_buffer.h"
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "led.h"
#include "system.h"
#include "unittest.h"


extern ring_buffer *t_buff;
extern ring_buffer *r_buff;


/*******UART0 INTIALIZATION FUNCTION-COMMON FOR BOTH MODES******************
 In this function we intialize the baud rate,ports and RX and TX for the UART0 mode.
 We set the NVIC for UART) interrupt mode.

*****************************************************************************/
void Init_UART0() {

        //set port a and uart 0 for use
        SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
        SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
        //set clock source for uart0-MCGPLLCLK
        SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);
        SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL_MASK;

        // Set pins to UART0 Rx and Tx
        PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Rx
        PORTA->PCR[2] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Tx
```

```
        //clear TE and RE before setting baud rate
        UART0->C2 &= ~ (UART0_C2_TE_MASK| UART0_C2_RE_MASK);

// Nested Vector Interrupt Controller (NVIC) module for interrupt mode of UART0 operation
#if UART_MODE == INTERRUPT_MODE
NVIC_EnableIRQ(UART0_IRQn);
NVIC_SetPriority(UART0_IRQn, 2);
#endif


        //UART0 set baud registers -48Mhz clock,115200 baud rate and  as oversampling
//sbr=48Mhz/(115200*16)=26
        UART0->BDH = 0x00;     //00
        UART0->BDL = 0x1A;     //26
        UART0->C4 = 0x0F;      //osr=16

        // 8-bit communication, no parity, one stop bit, LSB first, no inversion configuration

        UART0->C1 = 0x00;      //8bit,no parity check,even parity,no loop mode
        UART0->C3 = 0x00;
        UART0->MA1 = 0x00;
        UART0->MA2 = 0x00;     //
        UART0->S1 |= 0x1F;     //all errors
        UART0->S2 |= 0xC0;

#if UART_MODE == INTERRUPT_MODE
UART0->C2 = UART0_C2_RIE_MASK;
#endif
//enable transmitter and receiver
UART0->C2 |= UART0_C2_TE_MASK| UART0_C2_RE_MASK;

}

/*********************UART INTERRUPT MODE FUNCTIONS**********************
 In this file we have set of functions we use to transmit and receive characters
 using UART0 INTERRUPT MODE.
*************************************************************************/

//void UART0_Transmit_Poll(char data) {
//        while(!(UART0->S1&UART_S1_TDRE_MASK) && !(UART0->S1&UART_S1_TC_MASK));
//                UART0->D = data;
//}

/*********UART POLLING MODE FUNCTION-Checks for transmission**********************
 In this function the Initial condition of transmission i.e we use
 it to check if transmit is available.
*************************************************************************/
```

```
uint8_t UART0_check()
{
	while(!(UART0->S1&UART_S1_TDRE_MASK) && !(UART0->S1&UART_S1_TC_MASK));

	return 0;
}
/***UART POLLING MODE FUNCTION-Transmits data**********************************
 In this function we take a character and transmit data assuming transmit is available.
*******************************************************************************/
void UART0_poll_tx(char data) {
	//PRINTF("TRANSMITTING DATA\n \r");
	UART0->D = data;
}
/*********UART POLLING MODE FUNCTION-Transmits data***********************
 In this function we take a character in function and use the above functions to transmit data.
*******************************************************************************/
void UART0_poll_putchar(char data)			//tx
{
	init_LED();
	wait_transmit_led();
	if( UART0_check() == 0 )
	{
		UART0_poll_tx(data);

		//ring_status remove_buff = buff_remove_item(r_buff);
	}
}

//char UART0_Receive_Poll(void) {
//		while (!(UART0->S1 & UART0_S1_RDRF_MASK))
//			;
//		return UART0->D;
//}
/******UART POLLING MODE FUNCTION-Checks for receiving condition******************
 In this function the Initial condition of receiving data i.e we use
 it to check if receive is availabLe.
*******************************************************************************/
uint8_t UART0_rec_check()
{
	while(!(UART0->S1 & UART0_S1_RDRF_MASK));
		return 0;
}
```

```
/*********UART INTERRUPT MODE FUNCTION-Receives data**************************
 In this function we return a character assuming receive is available.
*************************************************************************/

char UART0_poll_rx()
{
        //PRINTF("Receiving DATA\n \r");
        return UART0->D;
}
/**********UART INTERRUPT MODE FUNCTION-Receives data**************************
 In this function we return character and use the above 2 functions to receive data.
*************************************************************************/

char UART0_poll_getchar()                       //rx
{
        init_LED();
        wait_receive_led();
        char rec_data;
                if( UART0_rec_check() == 0 )
                {
                        rec_data = UART0_poll_rx();
#if MODE == ECHO_MODE
                        ring_status add_buff = buff_add_item(r_buff, rec_data);
                        PRINTF("\n \r Add_buff status %d",add_buff);
//                      ring_status remove_buff = buff_remove_item(r_buff);
//                      PRINTF("\n\n\rremove_buff statusssss %d",remove_buff);
#endif
                }
                return rec_data;
}

/******************UART INTERRUPT MODE FUNCTION-Receives data******************
 In this function we get string from the user.
*************************************************************************/
void uart_getstr_poll(unsigned char *string)
//Receive a character until carriage return or newline
{
unsigned char i=0,a=0;
while((a!='\n') && (a!='\r'))
{

if( UART0_rec_check() == 0 )
{
*(string+i)= UART0_poll_rx();
}
```

```
UART0_poll_putchar(*(string+i));

a = *(string+i);
i++;
}

//i++;
*(string+i) = '\0';
putstr(string);
}

/*********************************************************************************
 *                                        PES PROJECT 5
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                  Cross Platform IDE: MCUXpresso IDE v11
 *                                        Cross-Compiler: ARM GCC
 *                                              uartpoll.h
 *********************************************************************************/
/*********************************REFERENCE*********************************
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf
*********************************************************************************/
#ifndef UARTPOLL_H_
#define UARTPOLL_H_

#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

#define POLLING_MODE 0
#define INTERRUPT_MODE 1

#define UART_MODE INTERRUPT_MODE //POLLING_MODE


//extern void uart0_isr(void);
#undef VECTOR_028
#define VECTOR_028 UART0_IRQHandler()

//void uart0_isr(void);
void Init_UART0();
void UART0_Transmit_Poll(char data);
uint8_t UART0_check();
void UART0_poll_tx(char data);
void UART0_poll_putchar(char data)     ;
```

```
char UART0_Receive_Poll(void);
uint8_t UART0_rec_check();
char UART0_poll_rx();
char UART0_poll_getchar();
void uart_getstr_poll(unsigned char *string);

#endif /* UARTPOLL_H_ */


/*********************************************************************************
 *                                      PES PROJECT 5
 *        AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                Cross Platform IDE: MCUXpresso IDE v11
 *                                      Cross-Compiler: ARM GCC
 *                                      uartinterrupt.c
*********************************************************************************/
 /**************References*****************************************************
  https://www.pantechsolutions.net/blog/how-to-receive-a-string-from-uart/
*********************************************************************************/
#include "uartinterrupt.h"
#include "uartpoll.h"
#include "cir_buffer.h"
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"

extern char str[100];
extern ring_buffer *r_buff;
ring_status       rx_status;

/**********************UART INTERRUPT MODE FUNCTIONS**********************
 In this file we have set of functions we use to transmit and receive characters
 using UART0 INTERRUPT MODE.
*********************************************************************************/
#if UART_MODE == INTERRUPT_MODE

//void UART0_Transmit_Poll(char data) {
//        while(!(UART0->S1&UART_S1_TDRE_MASK) && !(UART0->S1&UART_S1_TC_MASK));
//                UART0->D = data;
//}
/*******UART INTERRUPT MODE FUNCTION-Checks for transmission**********************
 In this function the Initial condition of transmission i.e we use
 it to check if transmit is availabe.
*********************************************************************************/
```

```
uint8_t UART0_int_check()
{
        while(!(UART0->S1&UART_S1_TDRE_MASK) && !(UART0->S1&UART_S1_TC_MASK));

        return 0;
}

/****UART INTERRUPT MODE FUNCTION-Transmits data*****************************
 In this function we take a character and transmit data assuming transmit is available.
*******************************************************************************
****/
void UART0_int_tx(char data) {
        //PRINTF("TRANSMITTING DATA\n \r");
        UART0->D = data;
}
/******UART INTERRUPT MODE FUNCTION-Transmits data*****************************
 In this function we take a character in function and use the above functions to transmit data.
*******************************************************************************/
void UART0_int_putchar(char data)                       //tx
{
        if( UART0_int_check() == 0 )
        {
                UART0_int_tx(data);
        }
}

//char UART0_Receive_Poll(void) {
//              while (!(UART0->S1 & UART0_S1_RDRF_MASK))
//                      ;
//              return UART0->D;
//}

/*********UART INTERRUPT MODE FUNCTION-Checks for receiving condition*************
 In this function the Initial condition of receiving data i.e we use
 it to check if receive is available.
*******************************************************************************/

uint8_t UART0_int_rec_check()
        {
                while(!(UART0->S1 & UART0_S1_RDRF_MASK));
                return 0;
        }

/*********UART INTERRUPT MODE FUNCTION-Receives data*****************************
 In this function we return a character assuming receive is available.
*******************************************************************************/
char UART0_int_rx()
```

```
{
        //PRINTF("Receiving DATA\n \r");
        return UART0->D;
}
/*****UART INTERRUPT MODE FUNCTION-Receives data************************
 In this function we return character and use the above 2 functions to receive data.
*************************************************************************/
char UART0_int_getchar()                        //rx
{
        char a;
                if( UART0_int_rec_check() == 0 )
                {
                        a=UART0_int_rx();
                }
                return a;
}

/******UART INTERRUPT MODE FUNCTION-Receives data************************
 In this function we get string from the user.
*********************************************************************/
void uart_getstr_int(unsigned char *string)  //Receive a character until carriage return or
newline
{
unsigned char i=0,a=0;
while((a!='\n') && (a!='\r'))
{
*(string+i)= UART0_int_getchar();
UART0_int_putchar(*(string+i));
a = *(string+i);
i++;
}

//i++;
*(string+i) = '\0';

}
/***************************UART INTERRUPT HANDLER**********************
 In this IRQ handler,when interrupt occurs then it jumps to this as service routine
 and executes the following:
 1.transmits data
 2.receives data
 3.In application mode,it gets string from user and when enter key is pressed
 it presents a report of all the characters entered to the terminal
 4.In echo mode,it stores data to buffer
 5.We use the start and end critical sections here.
*************************************************************************/
void UART0_IRQHandler()
{
```

```
//__disable_irq();
START_CRITICAL;
            char c;
#if MODE==APPLICATION_MODE
 PRINTF("\n \r APPLICATION MODE ON \n \r");
 uart_getstr_int(str);
 putstr(str);
 putstr("\n \n");
 app_mode(str);

#endif


  if (UART0->S1 & UART_S1_RDRF_MASK)
  {
  //c = UART0->D;
  init_LED();
  wait_receive_led();
  c=UART0_int_getchar();

  if (!(UART0->S1&UART_S1_TDRE_MASK) && !(UART0->S1&UART_S1_TC_MASK))
  {
  //UART0->D = c;
  init_LED();
  wait_transmit_led();
  UART0_int_putchar(c);
  }
  }

#if MODE==ECHO_MODE
    rx_status = buff_add_item(r_buff,c);
  PRINTF("\n\rRx2 status is: %d",rx_status);
  rx_status=buff_resize(r_buff,c);
#endif
 //__enable_irq();
END_CRITICAL;
}

#endif

/***************************UART FUNCTIONS*********************************
 This is the putstring condition in case we need to print strings without standard APIs.
**************************************************************************/

void putstr(unsigned char *string)
{

//unsigned char a=0;
```

```
while(*string){
//for(uint32_t i=0;i<=strlen(str);i++)

#if UART_MODE == INTERRUPT_MODE
UART0_int_putchar(*string++);
#endif

UART0_poll_putchar(*(string++));

//UART0_poll_putchar(*(string+strlen(str))) = '\0';
}
}
```

```
/*********************************************************************************
 *                                        PES PROJECT 5
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                                Cross Platform IDE: MCUXpresso IDE v11
 *                                        Cross-Compiler: ARM GCC
 *                                              uartinterrupt.h
 ********************************************************************/
/***********************************REFERENCE*********************************
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf

*****************************************************************************/
#ifndef UARTINTERRUPT_H_
#define UARTINTERRUPT_H_
#include<stdio.h>
#include <stdlib.h>
#include <stdint.h>

/*****************CRITICAL SECTION*********************/
#define START_CRITICAL __disable_irq()
#define END_CRITICAL  __enable_irq()
/*************************************************/

#if UART_MODE == INTERRUPT_MODE
char UART0_int_getchar();
uint8_t UART0_int_check();
void UART0_int_tx(char data);
void UART0_int_putchar(char data);
uint8_t UART0_int_rec_check();
char UART0_int_rx();
```

```
char UART0_int_getchar();
void UART0_IRQHandler();
void uart_getstr(unsigned char *string);
void uart_putstr(unsigned char *string);
void putstr(unsigned char *string);
void uart_getstr_int(unsigned char *string);
#endif
#endif /* UARTINTERRUPT_H_ */

/*************************************************************************
*                                    PES PROJECT 5
*   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 wait_receive_led(void)              //wait to receive data
{
        LED_RED_OFF();
        LED_GREEN_OFF();
        LED_BLUE_ON();
        delay(10);
}

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

void wait_transmit_led(void)    //when transmitting data
```

```
{
	LED_RED_OFF();
	LED_BLUE_OFF();
	LED_GREEN_ON();
	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 "uartinterrupt.h"
#include "uartpoll.h"
#include "fsl_debug_console.h"
#include "cir_buffer.h"
#include "unittest.h"
#include "system.h"
#include "application.h"


void init_LED(void);
void wait_receive_led(void);
void wait_transmit_led(void);
```

```
void delay(uint32_t d);
void error_led();
#endif

/*************************************************************************
 *                                          PES PROJECT 5
 *   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"

long int 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(){
START_CRITICAL;
timecount++;
END_CRITICAL;
}

uint8_t sec=0,min=0,hour=0;
extern long int timecount;

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

if(timer>600){
```

```
timer=0;
}
if(timer%10==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:%02lu \n ",hour,min,sec,timer);
}


/******************LOG LEVEL FUNCTIONS*********************************
                    PRINTS THE TYPE OF MODE USED
*************************************************************************/
void log_level(log_mode mode)
{
   if(mode == test)
   {
      putstr("\n\rMODE: Test");
   }
   else if(mode == debug)
   {
       putstr("\n\rMODE: Debug");
   }
   else if(mode == status)
   {
      putstr("\n\r MODE: Status");
   }
}
```

```
/******************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 Application mode"
                                                "\t Echo mode"
                                        };

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

```
                putstr(ch_arr[5]);

        }
        else if(func_nm == buffdestroy)
        {
                putstr("\t buff_destroy");
                putstr(ch_arr[6]);
        }
        else if(func_nm == buffresize)
        {
                putstr("\tbuff_resize");
                putstr(ch_arr[7]);
        }
        else if(func_nm == applicationmode)
        {
                putstr("\t app_mode");
                putstr(ch_arr[8]);
        }
        else
        {
                putstr("\t echo_mode");
                putstr(ch_arr[9]);
        }
}
/******************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 5
 *   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 "uartinterrupt.h"
#include "uartpoll.h"
#include "application.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
{
        test = 0,
        debug,
        status
}log_mode;

typedef enum
{
        buffinitialize = 0,
        buffcheck_full,
        buffcheck_empty,
        buffadd_item,
        buffremove_item,
        buffptr_valid,
        buffdestroy,
        buffresize,
```

```
        applicationmode,
        echomode
}log_func;


#define mode debug
//#define func INITIALIZE_buffer

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
```

```
/*************************************************************************
                                PES PROJECT 5
                    AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
                            Cross Platform IDE: MCUXpresso IDE v11
                           Cross-Compiler: ARM GCC
                                        system.c
*************************************************************************/
/*********************  References  *****************************************
https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-
microcontrollers/
 https://github.com/ucunit/ucunit
PES PROJECT 4
*************************************************************************/

#include "system.h"
#include "logger.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include <string.h>
#include "cir_buffer.h"
#include "uartinterrupt.h"
#include "uartpoll.h"
```

```c
#include "unittest.h"
/* Stub: Initialize your hardware here */
void System_Init(void)
{

	putstr("\n Initialization of system done\n ");
}

/* Stub: Shutdown your hardware here */
void System_Shutdown(void)
{
	putstr("\n Shutdowns the system");
	exit(0);
}

/* Stub: Recover your system from a safe state. */
void System_Recover(void)
{
	/* Stub: Recover the hardware */
	/* asm("\tRESET"); */
	putstr("\n System recovers.\n");
	exit(0);
}

/* Stub: Put system in a safe state */
void System_Safestate(void)
{
	putstr("\n Safe state of system\n");
	exit(0);
}

/* Stub: Write a string to the host/debugger/simulator */
void System_WriteString(char * string)
{
	putstr(string);
}

void System_WriteInt(int d)
{
	PRINTF(" %i", d);
}
```

/***

PES PROJECT 5

AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)

Cross Platform IDE: MCUXpresso IDE v11

Cross-Compiler: ARM GCC
system.h

**********References**********
https://mcuoneclipse.com/2018/08/26/tutorial-%CE%BCcunit-a-unit-test-framework-for-microcontrollers/
https://github.com/ucunit/ucunit
**********/

```
#ifndef SYSTEM_H_
#define SYSTEM_H_

/* function prototypes */
void System_Init(void);
void System_Shutdown(void);
void System_Safestate(void);
void System_Recover(void);
void System_WriteString(char * string);
void System_WriteInt(int d);

#endif /* SYSTEM_H_ */
```

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

/**********References**********
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf
http://cunit.sourceforge.net/screenshots.html
http://www.ucunit.org/
**********/

```
#include <unittest.h>

ring_buffer *buffer;
uint8_t *p;


uint8_t init_uctest(void)
{

        p = (ring_buffer*)malloc(sizeof(ring_buffer));
        uint8_t test=buff_initialize(p,8);
        UCUNIT_CheckIsNotNull(p);
        UCUNIT_CheckIsEqual(test,6);
        return 0;
```

```
}

uint8_t uctest_fill(){
        uint8_t test=buff_add_item(p,5);
        UCUNIT_CheckIsEqual(test,12);
        putstr("\n \r SUCCESSFUL ADDITION TO BUFFER");
        return 0;

}
uint8_t uctest_clear(void)
{
        free(p);
        putstr("\n \r CLEAR UCUNIT TEST BUFFER \n \r");
        return 0;
}


uint8_t uctest_overfill(){
        uint8_t test=buff_check_full(p);
        UCUNIT_CheckIsEqual(test,2);
        return 0;
}

uint8_t uctest_empty(){
        uint8_t test=buff_check_empty(p);
        UCUNIT_CheckIsEqual(test,4);
}

uint8_t uctest_resize(){
        uint8_t test=buff_resize(p);
        UCUNIT_CheckIsEqual(test,17);
        return 0;
}

void unit_test(void)
{

        UCUNIT_Init();
        init_uctest();
        uctest_fill();
        uctest_overfill();
        uctest_empty();
        uctest_resize();
        UCUNIT_WriteSummary();
        UCUNIT_TestcaseEnd();
        uctest_clear();

}
```

```
/*****************************************************************************
 *                                    PES PROJECT 5
 *   AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                              Cross Platform IDE: MCUXpresso IDE v11
 *                                    Cross-Compiler: ARM GCC
 *                                        unittest.h
 *****************************************************************************/
 /**************References*****************************************************
http://cache.freescale.com/files/32bit/doc/quick_ref_guide/KLQRUG.pdf
http://cunit.sourceforge.net/screenshots.html
http://www.ucunit.org/
https://github.com/ucunit/ucunit
*****************************************************************************/


#ifndef UNITTEST_H_
#define UNITTEST_H_

#include "logger.h"
#include "uartinterrupt.h"
#include "uartpoll.h"
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "cir_buffer.h"
```

```
#include "unittest.h"

void unit_test(void);
#define UCUNIT_WriteString(msg)    System_WriteString(msg)

/**
 * @Macro:      UCUNIT_WriteInt(n)
 *
 * @Description: Encapsulates a function which is called for
 *          writing an integer to the host computer.
 *
 * @param n:    Integer number which shall be written
 *
 * @Remarks:    Implement a function to write an integer to a host
 *          computer.
 *
 *          For most microcontrollers a special implementation of
 *          printf is available for writing to a serial
 *          device or network. In some cases you will have
 *          also to implement a putch(char c) function.
 */
#define UCUNIT_WriteInt(n)    System_WriteInt(n)

/**
 * @Macro:      UCUNIT_Safestate()
 *
 * @Description: Encapsulates a function which is called for
 *          putting the hardware to a safe state.
 *
 * @Remarks:    Implement a function to put your hardware into
 *          a safe state.
 *
 *          For example, imagine a motor controller
 *          application:
 *            1. Stop the motor
 *            2. Power brake
 *            3. Hold the brake
 *            4. Switch warning lamp on
 *            5. Wait for acknowledge
 *               ...
 *
 */
#define UCUNIT_Safestate()        System_Safestate()

/**
 * @Macro:      UCUNIT_Recover()
 *
 * @Description: Encapsulates a function which is called for
```

```
 *          recovering the hardware from a safe state.
 *
 * @Remarks:    Implement a function to recover your hardware from
 *          a safe state.
 *
 *          For example, imagine our motor controller
 *          application:
 *            1. Acknowledge the error with a key switch
 *            2. Switch warning lamp off
 *            3. Reboot
 *              ...
 *
 */
#define UCUNIT_Recover()        System_Reset()

/**
 * @Macro:      UCUNIT_Init()
 *
 * @Description: Encapsulates a function which is called for
 *          initializing the hardware.
 *
 * @Remarks:    Implement a function to initialize your microcontroller
 *          hardware. You need at least to initialize the
 *          communication device for transmitting your results to
 *          a host computer.
 *
 */
#define UCUNIT_Init()           System_Init()

/**
 * @Macro:      UCUNIT_Shutdown()
 *
 * @Description: Encapsulates a function which is called to
 *          stop the tests if a checklist fails.
 *
 * @Remarks:    Implement a function to stop the execution of the
 *          tests.
 *
 */
#define UCUNIT_Shutdown()       System_Shutdown()

/**
 * Verbose Mode.
 * UCUNIT_MODE_SILENT: Checks are performed silently.
 * UCUNIT_MODE_NORMAL: Only checks that fail are displayes
 * UCUNIT_MODE_VERBOSE: Passed and failed checks are displayed
 */
//#define UCUNIT_MODE_NORMAL
```

```
#define UCUNIT_MODE_VERBOSE

/**
 * Max. number of checkpoints. This may depend on your application
 * or limited by your RAM.
 */
#define UCUNIT_MAX_TRACEPOINTS 16

/************************************************************************/
/* **** End of customizing area *****                                  */
/************************************************************************/

/************************************************************************/
/* Some useful constants                                               */
/************************************************************************/

#define UCUNIT_VERSION "v1.0" /* Version info */

#ifndef NULL
#define NULL (void *)0
#endif

#ifndef TRUE
#define TRUE 1
#endif

#ifndef FALSE
#define FALSE 0
#endif

/* Action to take if check fails */
#define UCUNIT_ACTION_WARNING   0 /* Goes through the checks
                                     with message depending on level */
#define UCUNIT_ACTION_SHUTDOWN  1 /* Stops on the end of the checklist
                                     if any check has failed */
#define UCUNIT_ACTION_SAFESTATE 2 /* Goes in safe state if check fails */

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

/* Variables for simple statistics */
static int ucunit_checks_failed = 0; /* Numer of failed checks */
static int ucunit_checks_passed = 0; /* Number of passed checks */

static int ucunit_testcases_failed = 0; /* Number of failed test cases */
static int ucunit_testcases_passed = 0; /* Number of passed test cases */
static int ucunit_testcases_failed_checks = 0; /* Number of failed checks in a testcase */
```

```
static int ucunit_checklist_failed_checks = 0; /* Number of failed checks in a checklist */
static int ucunit_action = UCUNIT_ACTION_WARNING; /* Action to take if a check fails */
//static int ucunit_checkpoints[UCUNIT_MAX_TRACEPOINTS]; /* Max. number of tracepoints */
//static int ucunit_index = 0; /* Tracepoint index */

/***************************************************************************/
/* Internal (private) Macros                                               */
/***************************************************************************/

/**
 * @Macro:      UCUNIT_DefineToStringHelper(x)
 *
 * @Description: Helper macro for converting a define constant into
 *           a string.
 *
 * @Param x:    Define value to convert.
 *
 * @Remarks:    This macro is used by UCUNIT_DefineToString().
 *
 */
#define UCUNIT_DefineToStringHelper(x)    #x

/**
 * @Macro:      UCUNIT_DefineToString(x)
 *
 * @Description: Converts a define constant into a string.
 *
 * @Param x:    Define value to convert.
 *
 * @Remarks:    This macro uses UCUNIT_DefineToStringHelper().
 *
 */
#define UCUNIT_DefineToString(x)   UCUNIT_DefineToStringHelper(x)

#ifdef UCUNIT_MODE_VERBOSE
/**
 * @Macro:      UCUNIT_WritePassedMsg(msg, args)
 *
 * @Description: Writes a message that check has passed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *           Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:    This macro is used by UCUNIT_Check(). A message will
 *           only be written if verbose mode is set
 *           to UCUNIT_MODE_VERBOSE.
 *
```

```
 */
#define UCUNIT_WritePassedMsg(msg, args)                    \
  do                                                        \
  {                                                         \
    UCUNIT_WriteString(__FILE__);                           \
    UCUNIT_WriteString("\t :");                             \
    UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
    UCUNIT_WriteString("\t: passed:");                      \
    UCUNIT_WriteString(msg);                                \
    UCUNIT_WriteString("(");                                \
    UCUNIT_WriteString(args);                               \
    UCUNIT_WriteString(")\n");                              \
  } while(0)
#else
#define UCUNIT_WritePassedMsg(msg, args)
#endif

#ifdef UCUNIT_MODE_SILENT
#define UCUNIT_WriteFailedMsg(msg, args)
#else
/**
 * @Macro:      UCUNIT_WriteFailedMsg(msg, args)
 *
 * @Description: Writes a message that check has failed.
 *
 * @Param msg:   Message to write. This is the name of the called
 *           Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:    This macro is used by UCUNIT_Check(). A message will
 *           only be written if verbose mode is set
 *           to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_WriteFailedMsg(msg, args)                    \
  do                                                        \
  {                    \
        UCUNIT_WriteString("\n");
\
    UCUNIT_WriteString(__FILE__);                           \
    UCUNIT_WriteString("\t:");                              \
    UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
    UCUNIT_WriteString("\t: failed:");                      \
    UCUNIT_WriteString(msg);                                \
    UCUNIT_WriteString("(");                                \
    UCUNIT_WriteString(args);                               \
    UCUNIT_WriteString(")\n");                              \
  } while(0)
```

```
#endif

/**
 * @Macro:      UCUNIT_FailCheck(msg, args)
 *
 * @Description: Fails a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *           Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:    This macro is used by UCUNIT_Check(). A message will
 *           only be written if verbose mode is set
 *           to UCUNIT_MODE_NORMAL and UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_FailCheck(msg, args)              \
  do                                  \
  {                                   \
    if (UCUNIT_ACTION_SAFESTATE==ucunit_action)  \
    {                                 \
      UCUNIT_Safestate();             \
    }                                 \
    UCUNIT_WriteFailedMsg(msg, args);         \
    ucunit_checks_failed++;               \
    ucunit_checklist_failed_checks++;         \
  } while(0)

/**
 * @Macro:      UCUNIT_PassCheck(msg, args)
 *
 * @Description: Passes a check.
 *
 * @Param msg:   Message to write. This is the name of the called
 *           Check, without the substring UCUNIT_Check.
 * @Param args:  Argument list as string.
 *
 * @Remarks:    This macro is used by UCUNIT_Check(). A message will
 *           only be written if verbose mode is set
 *           to UCUNIT_MODE_VERBOSE.
 *
 */
#define UCUNIT_PassCheck(message, args)          \
  do                                  \
  {                                   \
    UCUNIT_WritePassedMsg(message, args);      \
    ucunit_checks_passed++;               \
  } while(0)
```

```
/*****************************************************************************/
/* Checklist Macros                                                          */
/*****************************************************************************/

/**
 * @Macro:      UCUNIT_ChecklistBegin(action)
 *
 * @Description: Begin of a checklist. You have to tell what action
 *           shall be taken if a check fails.
 *
 * @Param action: Action to take. This can be:
 *           * UCUNIT_ACTION_WARNING:   A warning message will be printed
 *                                      that a check has failed
 *           * UCUNIT_ACTION_SHUTDOWN:  The system will shutdown at
 *                                      the end of the checklist.
 *           * UCUNIT_ACTION_SAFESTATE: The system goes into the safe state
 *                                      on the first failed check.
 * @Remarks:    A checklist must be finished with UCUNIT_ChecklistEnd()
 *
 */
#define UCUNIT_ChecklistBegin(action)                   \
  do                                                    \
  {                                                     \
    ucunit_action = action;                             \
    ucunit_checklist_failed_checks = 0;                 \
  } while (0)

/**
 * @Macro:      UCUNIT_ChecklistEnd()
 *
 * @Description: End of a checklist. If the action was UCUNIT_ACTION_SHUTDOWN
 *           the system will shutdown.
 *
 * @Remarks:    A checklist must begin with UCUNIT_ChecklistBegin(action)
 *
 */
#define UCUNIT_ChecklistEnd()                           \
  if (ucunit_checklist_failed_checks!=0)                \
  {                                                     \
    UCUNIT_WriteFailedMsg("Checklist","");              \
    if (UCUNIT_ACTION_SHUTDOWN==ucunit_action)          \
    {                                                   \
      UCUNIT_Shutdown();                                \
    }                                                   \
  }                                                     \
  else                                                  \
  {                                                     \
```

```
    UCUNIT_WritePassedMsg("Checklist","");       \
  }

/*****************************************************************************/
/* Check Macros                                                              */
/*****************************************************************************/

/**
 * @Macro:      UCUNIT_Check(condition, msg, args)
 *
 * @Description: Checks a condition and prints a message.
 *
 * @Param msg:   Message to write.
 * @Param args:  Argument list as string
 *
 * @Remarks:    Basic check. This macro is used by all higher level checks.
 *
 */
#define UCUNIT_Check(condition, msg, args)          \
  if ( (condition) ) { UCUNIT_PassCheck(msg, args); } else { UCUNIT_FailCheck(msg, args); }

/**
 * @Macro:      UCUNIT_CheckIsEqual(expected,actual)
 *
 * @Description: Checks that actual value equals the expected value.
 *
 * @Param expected: Expected value.
 * @Param actual: Actual value.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsEqual(expected,actual)       \
  UCUNIT_Check( (expected) == (actual), "IsEqual \n \r", #expected "," #actual )

/**
 * @Macro:      UCUNIT_CheckIsNull(pointer)
 *
 * @Description: Checks that a pointer is NULL.
 *
 * @Param pointer: Pointer to check.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsNull(pointer)             \
  UCUNIT_Check( (pointer) == NULL, "\n \r IsNull", #pointer)
```

```
        /**
         * @Macro:      UCUNIT_CheckIsNotNull(pointer)
         *
         * @Description: Checks that a pointer is not NULL.
         *
         * @Param pointer: Pointer to check.
         *
         * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
         *
         */
        #define UCUNIT_CheckIsNotNull(pointer)            \
          UCUNIT_Check( (pointer) != NULL, "IsNotNull", #pointer)

        /**
         * @Macro:      UCUNIT_CheckIsInRange(value, lower, upper)
         *
         * @Description: Checks if a value is between lower and upper bounds (inclusive)
         *           Mathematical: lower <= value <= upper
         *
         * @Param value: Value to check.
         * @Param lower: Lower bound.
         * @Param upper: Upper bound.
         *
         * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
         *
         */
        #define UCUNIT_CheckIsInRange(value, lower, upper)   \
          UCUNIT_Check( ( (value>=lower) && (value<=upper) ), "IsInRange", #value "," #lower "," #upper)

        /**
         * @Macro:      UCUNIT_CheckIs8Bit(value)
         *
         * @Description: Checks if a value fits into 8-bit.
         *
         * @Param value: Value to check.
         *
         * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
         *
         */
        #define UCUNIT_CheckIs8Bit(value)                 \
          UCUNIT_Check( value==(value & 0xFF), "Is8Bit\n \r", #value )

        /**
         * @Macro:      UCUNIT_CheckIs16Bit(value)
         *
         * @Description: Checks if a value fits into 16-bit.
         *
```

```
 * @Param value: Value to check.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs16Bit(value)                  \
  UCUNIT_Check( value==(value & 0xFFFF), "Is16Bit \n \r", #value )

/**
 * @Macro:      UCUNIT_CheckIs32Bit(value)
 *
 * @Description: Checks if a value fits into 32-bit.
 *
 * @Param value: Value to check.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIs32Bit(value)                  \
  UCUNIT_Check( value==(value & 0xFFFFFFFF), "Is32Bit \n \r", #value )

/**
 * Checks if bit is set
 */
/**
 * @Macro:      UCUNIT_CheckIsBitSet(value, bitno)
 *
 * @Description: Checks if a bit is set in value.
 *
 * @Param value: Value to check.
 * @Param bitno: Bit number. The least significant bit is 0.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
 */
#define UCUNIT_CheckIsBitSet(value, bitno) \
  UCUNIT_Check( (1==(((value)>>(bitno)) & 0x01) ), "IsBitSet", #value "," #bitno)

/**
 * @Macro:      UCUNIT_CheckIsBitClear(value, bitno)
 *
 * @Description: Checks if a bit is not set in value.
 *
 * @Param value: Value to check.
 * @Param bitno: Bit number. The least significant bit is 0.
 *
 * @Remarks:    This macro uses UCUNIT_Check(condition, msg, args).
 *
```

```
 */
#define UCUNIT_CheckIsBitClear(value, bitno) \
  UCUNIT_Check( (0==(((value)>>(bitno)) & 0x01) ), "IsBitClear", #value "," #bitno)

/*****************************************************************************/
/* Testcases */
/*****************************************************************************/

/**
 * @Macro:      UCUNIT_TestcaseBegin(name)
 *
 * @Description: Marks the beginning of a test case and resets
 *           the test case statistic.
 *
 * @Param name:  Name of the test case.
 *
 * @Remarks:    This macro uses UCUNIT_WriteString(msg) to print the name.
 *
 */
        \

/**
 * @Macro:      UCUNIT_TestcaseEnd()
 *
 * @Description: Marks the end of a test case and calculates
 *           the test case statistics.
 *
 * @Remarks:    This macro uses UCUNIT_WriteString(msg) to print the result.
 *
 */
#define UCUNIT_TestcaseEnd()                            \
  do                                                    \
  {                                                     \
    UCUNIT_WriteString("\n \r ====================================\n \r");  \
    if( 0==(ucunit_testcases_failed_checks - ucunit_checks_failed) ) \
    {                                                   \
      UCUNIT_WriteString("\n \r Testcase passed.\n");           \
      ucunit_testcases_passed++;                        \
    }                                                   \
    else                                                \
    {                                                   \
      UCUNIT_WriteFailedMsg("\n \r EndTestcase","");            \
      ucunit_testcases_failed++;                        \
    }                                                   \
    UCUNIT_WriteString("\n \r====================================\n \r"); \
  }                                                     \
  while(0)
```

```
/**
 * @Macro:      UCUNIT_Tracepoint(index)
 *
 * @Description: Marks a trace point.
 *          If a trace point is executed, its coverage state switches
 *          from 0 to the line number.
 *          If a trace point was never executed, the state
 *          remains 0.
 *
 * @Param index: Index of the tracepoint.
 *
 * @Remarks:    This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_Tracepoint(index)                    \
  if(index<UCUNIT_MAX_TRACEPOINTS)                  \
  {                                                 \
    ucunit_checkpoints[index] = __LINE__;           \
  }                                                 \
  else                                              \
  {                                                 \
    UCUNIT_WriteFailedMsg("Tracepoint index", #index);    \
  }

/**
 * @Macro:      UCUNIT_ResetTracepointCoverage()
 *
 * @Description: Resets the trace point coverage state to 0.
 *
 * @Param index: Index of the trace point.
 *
 * @Remarks:    This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_ResetTracepointCoverage()                \
  for (ucunit_index=0; ucunit_index<UCUNIT_MAX_TRACEPOINTS; ucunit_index++) \
  {                                                     \
    ucunit_checkpoints[ucunit_index]=0;         \
  }

/**
 * @Macro:      UCUNIT_CheckTracepointCoverage(index)
 *
 * @Description: Checks if a trace point was covered.
 *
 * @Param index: Index of the trace point.
```

```
 *
 * @Remarks:    This macro fails if index>UCUNIT_MAX_TRACEPOINTS.
 *
 */
#define UCUNIT_CheckTracepointCoverage(index)    \
   UCUNIT_Check( (ucunit_checkpoints[index]!=0), "TracepointCoverage", #index);

/*****************************************************************************/
/* Testsuite Summary                                                         */
/*****************************************************************************/

/**
 * @Macro:      UCUNIT_WriteSummary()
 *
 * @Description: Writes the test suite summary.
 *
 * @Remarks:    This macro uses UCUNIT_WriteString(msg) and
 *              UCUNIT_WriteInt(n) to write the summary.
 *
 */
#define UCUNIT_WriteSummary()                                   \
{                                                               \
   UCUNIT_WriteString("\n \r ***************************************");   \
   UCUNIT_WriteString("\n \r Testcases: failed: ");                 \
   UCUNIT_WriteInt(ucunit_testcases_failed);                        \
   UCUNIT_WriteString("\n \r          passed: ");                   \
   UCUNIT_WriteInt(ucunit_testcases_passed);                        \
   UCUNIT_WriteString("\n \r Checks:    failed: ");                 \
   UCUNIT_WriteInt(ucunit_checks_failed);                           \
   UCUNIT_WriteString("\n \r        passed: ");                     \
   UCUNIT_WriteInt(ucunit_checks_passed);                           \
   UCUNIT_WriteString("\n \r ***************************************\n"); \
}

#endif /*UCUNIT_H_*/

/*****************************************************************************
*                                    PES PROJECT 5
*              AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
*
*                        Cross Platform IDE: MCUXpresso IDE v11
*                                Cross-Compiler: ARM GCC
*
                                    application.c
 *****************************************************************************/
/*********************************REFERENCE*********************************
https://www.programmingsimplified.com/c-program-find-characters-frequency
*****************************************************************************/
```

```c
#include "application.h"


#if MODE == APPLICATION_MODE

/***************************APPLICATION MODE FUNCTION**********************
 In this function we take a string from the user and when ENTER key is pressed
 we get desired report of number of entries of characters from  A to Z,a to z
 or 0-9 made to the system.
*****************************************************************************/

void app_mode(char string[])
{

  int a = 0, count[26] = {0},count2[26] = {0},count3[26] = {0}, x;

  while (string[a] != '\0')            //terminating condition
  {

    if (string[a] >= 'a' && string[a] <= 'z')
    {
      x = string[a] - 'a';
      count[x]++;
    }
    else if (string[a] >= 'A' && string[a] <= 'Z')
    {
      int y = string[a] - 'A';
      count2[y]++;
    }
    else if (string[a] >= '0' && string[a] <= '9')
    {
      int z = string[a] - '0';
      count3[z]++;
    }

    a++;
  }

  for (a = 0; a < 26; a++)
  {
            if(count[a] != 0)
            {
      printf("\n \r %c -> %d ", a + 'a', count[a]);
            }
  }
  for (int d = 0; d < 26; d++)
  {
```

```
            if(count2[d] != 0)
            {
        printf("\n \r %c -> %d ", d + 'A', count2[d]);
            }
  }
  for (int c = 0; c < 10; c++)
  {
            if(count3[c] != 0)
            {
        printf("\n \r %c -> %d ", c + '0', count3[c]);
            }
  }

}

#endif

/*************************************************************************
 *                                PES PROJECT 5
 *                AAKSHA JAYWANT (AAJA1276) & RUCHA BORWANKAR (RUBO1268)
 *                    Cross Platform IDE: MCUXpresso IDE v11
 *                              Cross-Compiler: ARM GCC
 *                                application.h
 *************************************************************************/
/*********************************REFERENCE*********************************
https://www.programmingsimplified.com/c-program-find-characters-frequency
*************************************************************************/
#ifndef APPLICATION_H_
#define APPLICATION_H_

#define ECHO_MODE 0
#define APPLICATION_MODE 1

#include "uartpoll.h"
#include "logger.h"
#include "uartpoll.h"
#include "uartinterrupt.h"
#include <stdio.h>
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "cir_buffer.h"
#include "led.h"
#include "unittest.h"
#include "system.h"
```

```
#define MODE APPLICATION_MODE//ECHO_MODE//

void app_mode(char string[]);

#endif /* APPLICATION_H_ */
```

```
/*************************************************************************
 *                                PES PROJECT 5
 *   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;

/*************************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 = (char*) malloc(sizeof(char)*capacity);
		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,char 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){
				putstr("\n Wrap around occurs");
				}
				p->head = 0;
				p->count++;
				p->head_count++;
				return wrap_around;
			}
			else if(buff_check_full(p) == buffer_full)
			{
				init_LED();
				error_led();
				if(mode==test || mode==debug)
				{
				putstr("\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("*************************************************************");
				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");
                        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)
                        {
                              putstr("\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)
                           {
                                  putstr("\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
{
      putstr("\n********* wait************ \n");
      //PRINTF("waitttttttttttttt");
}
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 ***************************************************");
		putstr("\n \r Buffer is resized");

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

		return memory_reallocated;
	}

	else
	{
		return memory_not_reallocated;
	}
}
```

```
/*****************************************************************************
 *                              PES PROJECT 5
 *   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{
    char *buffer;
    uint8_t head;
    uint8_t tail;
    uint8_t capacity;
    uint8_t count;
    uint8_t head_count;
    uint8_t tail_count;
    char *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,char 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
```