PROJECT 5 FINAL PDF

Readme For project 5 Team Members :

1)Suraj Thite (suraj.thite@colorado.edu)

2)Atharv Desai (atharv.desai@colorado.edu)

This is a readme file for the fifth project assignment in the Principles of Embedded Software Course for FALL '19.

The below enumerated files are contained in the repository

1) PES_Project5

i) Project Settings

ii) Build Targets

iii) Binaries

iv) Includes

v) CMSIS

vi) Board

vii) Drivers

viii) Source a) main.c & main.h

b) logger.c & logger.h

c) RGBled.c & RGBled.h

d) char_count_update.c & char_count_update.h

e) unittest.c & unittest.h

f) circularbuff.c & circularbuff.h

g) timestamp.c & timestamp.h

h) uart_interrupt.c & uart_interrupt.h

ix) Startup

x) Utilities

xi) Debug

xii) PESProject5 PE Debug.launch

2) Readme File (Readme Markdown File)

3) UART Transactions Screenshot

INSTALLATION & EXECUTION NOTES

The code is tested on the enviornment below:

* MCUXpresso IDE which is an easy to use Eclipse-based development environment for NXP® MCUs based on Arm® Cortex®-M cores.

* During this project, this IDE was used to code,execute circular buffer functions and interface them with polling and interrupt based UART of FRDM board on FRDM-KL25z and print their output on the IDE's serial terminal.

* Also, we were able to check and verify the workability of the UART and Buffer using the unittest cases in the project.

* Using Debug Port Logic Analyser, the PTA1 and PTA2 pin data in the form of waveform was used to capture UART transactions

* The hardware used in this project was FRDM-KL25Z board which has been designed by NXP and built on ARM® Cortex™-M0+ processor.

* The editor used to build the code is gedit version 2.3.8.1 on Linux Mint Machine.

* To execute the executable file simply type ./(filename) in linux gcc environment while click on debug (bug icon) and then resume button to execute the file on MCUXpresso.

* Kindly use notepad++ for viewing .out files ,particularly for first output since they have been misaligned due to character "Space or Tab" encoding.

* Set #define to 1 (ECHO) or 0 (APPLICATION) respectively.

* Set modes to Test, Debug or Status mode by setting the vale for variable 'a' in logger.c file accordingly.

DIFFICULTIES & OBSERVATIONS:

* While capturing the UART transactions output on Logic Analyser, faced issues in capturing the UART TX and RX waveforms in timing mode. Also, we learned to interpret the data on those lines in binary format.

* While designing the circular buffer functions, we were not able to reallocate a new buffer while retaining the old data and adding new data further.

* While implementing the UART communication with specific baud rate, performed calculations such as calculating reload value for 10Hz frequency and setting systick timer frequency for the same.

* While implementing logger earlier, the enum values were being passed as arguments in integer format only. But on accessing them in other .c files using extern keyword, the issue was resolved. Also, integrated the timestamp with the logger and segregated the count value between hours, minutes and seconds to get the timestamp in appropriate format.

FINAL CODE

```c
/*
 * main.h
 *
 *  Created on: Nov 9, 2019
 *      Author: SURAJ THITE & ATHARV DESAI
 */

#ifndef MAIN_H_
#define MAIN_H_

#include "circularbuff.h"
#include "char_count_update.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 "circularbuff.h"
#include "uart_interrrupt.h"
#include "time_stamp.h"
#include "logger.h"

void echo_function_interrupt();
void echo_function_poll(char a);
void application_poll(uint8_t *ch);
void application_int();

#endif /* MAIN_H_ */
```

/**
 * @file   PES_ASSIGNMENT5.c
 * @brief   Application entry point.
 */
/* TODO: insert other include files here. */
#include "main.h"
#include "unitTest.h"
#include "RGBled.h"

//Program Execution Control Variables
#define ECHO 0   //Enable Echo mode of Operation
#define APPLICATION 0   //Enable Application mode of Operation
#define TESTMODE 1   //Enable Unit Testing Mode

// Program Definitions

#define buff_length (15)   //Set Buffer Size
#define BAUDRATE (9600)   //Set Baud for UART communications

/* TODO: insert other Global declarations here. */

uint8_t no_of_blocks=0;
cbuff *rx;
uint8_t *element_deleted;
uint8_t* info;
int tx_flag=1;
int count=0;

```
char recv;
bool rx_flag;
bool rx_flag_1;
extern uint8_t char_count[256];
bool int_exit;
extern modes a;                          //Modes for logging

/*
 * @brief   Application entry point.
 */
int main(void) {

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

	Init_SysTick();            //Initialize the Systick Timer

	RGB_init();   //Initialize the LEDs

	led_switch(0);       //Initialize the LED to Blue

	rx= malloc(sizeof(cbuff));  //Initialize the Rx circular buffer with size of
structure

	rx->cbuffptr = malloc(sizeof(int8_t) * buff_length);   //store the starting
address of the Length in the base pointer of the structure

	cbuff_init(rx,buff_length);       //Initialize circular buffer with length =15

	Init_UART0(BAUDRATE*2);                  //Initialize UART
//Unit Testing
#if TESTMODE
	UCUNIT_Init();
	UCUNIT_WriteString("\n**************************************");
	UCUNIT_WriteString("\nuCUnit demo application");
	UCUNIT_WriteString("\nDate:     ");
	UCUNIT_WriteString(__DATE__);
	UCUNIT_WriteString("\nTime:     ");
	UCUNIT_WriteString(__TIME__);

	UCUNIT_WriteString("\n**************************************");
	Testsuite_RunTests();
	UCUNIT_Shutdown();

#else
//ECHO mode of Operation
#if ECHO
#if USE_UART_INTERRUPTS
	if(a==1 || a==2)
		Log_String(a,Main,"**** ECHO USING INTERRUPTS****");
	while (1)
	{
		echo_function_interrupt();  //Implementing Interrupts
	}
```

```
#else
	if(a==1 || a==2)
		Log_String(a,Main,"**** ECHO USING POLLING****");
	while (1)
	{
		char a = uart_rx();  //Receive a character from receive polling
		echo_function_poll(a);	//Implementing Polling
	}

#endif
#endif

#if APPLICATION			//Application Mode of Operation
#if USE_UART_INTERRUPTS
	if(a==1 || a==2)
		Log_String(a,Main,"**** APPLICATION USING INTERRUPTS****");   //1
	while (1)
	{
		application_int();		//Application using Interrupts
	}

#else				//Application using polling
	if(a==1 || a==2)
		Log_String(a,Main,"**** APPLICATION USING POLLING****"); //1
	while(1)
	{
		char a = uart_rx();
		if(a == '.')			//Generate report on reception of null character
		{
			if(a==1 || a==2)
				Log_String(a,Main,"****Generating Report*****");//Generate
report
			generate_report();

			//Clear the array where previous values were stored
			for (int i=65;i<=90;i++)
			{
				char_count[i] =0;
			}
			for (int i=97;i<=122;i++)
			{
				char_count[i] =0;
			}
			//	break;
		}
		application_poll(&a);			//Application poll
	}

#endif
#endif

	return 0 ;
#endif
}

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

```
 * Function Name:void echo_function_interrupt()
 * Description :This function echoes the characters recieved from the sender via non-
blocking mode of operation
 * @input: char
 * @Return : void

*****************************************************************************************
**************/
void echo_function_interrupt()
{
	if(rx_flag ==1)			//Check for rx flag ie set in IRQ handler
	{
		if(a==1 || a==2)
			Log_String(a,Echo_function_interrupt,"Character Received");	//1
		uint8_t *current = rx->head;			//Access memory location of the
circular buffer ->head
		current --;  //Since head points to next empty memory location, decrement
current to access last value stored
		char time_buf[2048] = {0};
		sprintf(time_buf, " \n %c \n", *current);
		UART0_print_string(time_buf);			//Transmit character value via
UART
		if(a==1 || a==2)
			Log_String(a,Echo_function_interrupt,"Character Transmitted");//1
		rx_flag=0; //Clear Rx flag
	}
}

/*****************************************************************************************
***************
 * Function Name:void echo_function_poll(char a)
 * Description :This function echoes the characters recieved from the sender via blocking
mode of operation
 * @input: char
 * @Return : void

*****************************************************************************************
**************/
void echo_function_poll(char a)
{
	if(rx_flag_1==1)	//Check for Rx_flag_1 set upon reception of rx signal in
receive wait state
	{
		if(a==1 || a==2)
			Log_String(a,Echo_function_poll,"Character Received");//1
		rx_flag_1=0;  //Clear the flag
		uart_tx(a);   //Echo the value to the terminal screen
		if(a==1 || a==2)
			Log_String(2,Echo_function_poll,"Character Transmitted");//1
	}
}
/*****************************************************************************************
***************
 * Function Name:void application_poll(uint8_t *ch)
 * Description :This function runs the application mode of operation in polling mode
 * @input: pointer to a char value.
 * @Return : void
```

```
**********************************************************************************
**************/
void application_poll(uint8_t *ch)
{
	if(rx_flag_1==1)	//Check for any value recieved
	{
		if(a==1 || a==2)
			Log_String(a,Application_poll,"Character Received"); //1
		rx_flag_1=0;  //Clear the flag
		if(a==1 || a==2)
			Log_String(a,charactercount,"Character Count Incremented");   //1
		character_count(ch); //increment the character count for report generation
		//printf("%d",char_count[51]);
	}
}

/**********************************************************************************
***************
 * Function Name:void application_int()
 * Description :This function runs the application mode of operation in interrupt mode
 * @input: void
 * @Return : void

**********************************************************************************
**************/
void application_int()
{
	if(rx_flag ==1)			//Check for rx_flag
	{
		if(a==1 || a==2)
			Log_String(a,Application_int,"Character Received");//1
		uint8_t *current = rx->head;	//Store the head value in temporary
pointer
		current --;   //Point current to the memory location where the last char
value was stored
		if(*current == '.')  //Check for reception of '.' character
		{

			if(a==1 || a==2)
				Log_String(a,Main,"****Generating Report*****");//1
			generate_report();   //Generate Report

			//Clear the values where all previous values were stored
			for (int i=65;i<=90;i++)
			{
				char_count[i] =0;
			}
			for (int i=97;i<=122;i++)
			{
				char_count[i] =0;
			}

		}
		if(a==1 || a==2)
			Log_String(a,charactercount,"Character Count Incremented"); //1
		character_count(current); //Increase the character count for report
generation
```

```
                rx_flag=0;
        }
}


/*
 * char_count_update.h
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE & Atharv Desai
 */

#ifndef CHAR_COUNT_UPDATE_H_
#define CHAR_COUNT_UPDATE_H_
#include "main.h"
uint8_t* character_count(uint8_t *char_val);
void generate_report();

#endif /* CHAR_COUNT_UPDATE_H_ */
```

```
/*
 * char_count_update.c
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE & ATHARV DESAI
 */

#include "char_count_update.h"

uint8_t char_count[256];

/*****************************************************************************************
***************
 * Function Name:uint8_t* character_count(uint8_t *char_val)
 * Description :This function receives the character and updates its count in the
char_count array
 * @input:  pointer to uint8_t
 * @Return : pointer

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

uint8_t* character_count(uint8_t *char_val)
{
        char_count[*char_val] ++;   //Increment character value in the char_count global
variable
        return char_count;   //Return pointer to array
}

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

```
 * Function Name: void generate_report()
 * Description : This function generates report to be printed on detection of '.'
character
 * @input:  void
 * @Return : void

*****************************************************************************************
**************/
void generate_report()
{
	printf("\n \r");
	for (int i=65;i<=90;i++)    //Check for A-Z and a-z ascii characters only
	{
		if(char_count[i]!=0) //If count is not zero , print the value
		{
			printf("%c-%d,",i,char_count[i]);
		}
	}

	for (int i=97;i<=122;i++)
		{
			if(char_count[i]!=0)
			{
				printf("%c-%d,",i,char_count[i]);
			}
		}
}
```

```
/*
 * circularbuff.h
 *
 *  Created on: Nov 9, 2019
 *      Author: SURAJ THITE &  ATHARV DESAI
 */

#ifndef CIRCULARBUFF_H_
#define CIRCULARBUFF_H_

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include "main.h"
typedef struct
{
	uint8_t *cbuffptr;
    uint8_t *newcbuffptr;
    uint8_t *head;
    uint8_t *tail;
    uint16_t size;
    uint8_t count;
}cbuff;
```

```c
typedef enum
{
    cbuff_init_success,
    cbuff_init_fail,
    cbuff_empty,
       wrap_add,
       wrap_remove,
    cbuff_not_empty,
    cbuff_full,
    cbuff_not_full,
    null_ptr,
    buffer_NA,
    cbuff_success,
       buffer_init_failed,
       buffer_init_success,
       ptr_valid,
       ptr_invalid,
       destroy_failed,
       destroy_pass
}cbuff_status;

cbuff_status cbuff_init(cbuff *ptr, uint16_t length);
cbuff_status cbuff_add(cbuff *ptr, uint8_t val);
cbuff_status cbuff_delete(cbuff *ptr, uint8_t *val);
cbuff_status cbuff_isempty(cbuff *ptr);
cbuff_status cbuff_check_full(cbuff *ptr);
cbuff_status cbuff_resize(cbuff *ptr,uint8_t length);
void cbuff_print(cbuff* ptr);
cbuff_status verify_ptr(uint8_t *ptr1,cbuff *ptr);
cbuff_status verify_init(cbuff* ptr);
cbuff_status cbuff_destroy(cbuff* ptr);

#endif /* CIRCULARBUFF_H_ */
```

```c
/*
 * circularbuff.c
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE & ATHARV DESAI
 */
#include "circularbuff.h"
#include "fsl_debug_console.h"
#include "RGBled.h"
extern modes a;
```

```
/*****************************************************************************
******************
 * Function Name:cbuff_status cbuff_init(cbuff *ptr, uint16_t length)
 * Description :This Function Initializes the circular buffer
 * @input: pointer to circular buffer and length of the circular buffer
 * @Return : error status messages

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

cbuff_status cbuff_init(cbuff *ptr, uint16_t length)
{

	if(ptr==NULL || length <=0)
	{
		led_switch(1);	//Change LED to red upon error
		return null_ptr;
	}
	else
	{
		if((ptr->cbuffptr)==NULL)
		{
			led_switch(1);	//Change LED to red upon error
			ptr->head=NULL;	//Initialize  head pointer to NULL
			ptr->tail=NULL;	//Initialize tail pointer to NULL
			ptr->count=0;//Initialize count to zero
			ptr->size=0; //Initialize size to zero
			return cbuff_init_fail;	//Return Fail status message
		}
		else
		{
			ptr->cbuffptr= (uint8_t*)malloc(sizeof(uint8_t)*length);
	//Allocate memory for the circular buffer
```

```
                    ptr->size= length;  //Set size of the buffer
                    ptr->count=0;//Initialize count to zero
                    ptr->head=ptr->cbuffptr;  //Set head to base address
                    ptr->tail=ptr->cbuffptr;  //Set tail to base address
                    return  cbuff_init_success;      //Return status
             }
      }
}

/**************************************************************************************************
 * Function Name:cbuff_status cbuff_check_full(cbuff *ptr)
 * Description :This Function checks whether circular buffer is full or not
 * @input: pointer to circular buffer
 * @Return : error status messages

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

cbuff_status cbuff_check_full(cbuff *ptr)
{
      if(ptr==NULL)
      {
             led_switch(1);            //Change LED to red upon error
             return null_ptr;    //Return Status
      }
      else if ((ptr->cbuffptr)==NULL)
      {
             return buffer_NA;   //Return Status
      }
      else if ((ptr->count)==(ptr->size))
      {
```

```
		led_switch(1);			//Change LED to red upon error
		return cbuff_full;  //Return Status
	}
	else
	{
		return cbuff_not_full; //Return Status
	}
}

/*****************************************************************************
******************
 * Function Name:cbuff_status cbuff_isempty(cbuff *ptr)
 * Description :This Function checks whether circular buffer is empty or not
 * @input: pointer to circular buffer
 * @Return : error status messages

*****************************************************************************
******************/
cbuff_status cbuff_isempty(cbuff *ptr)
{
	if(ptr==NULL)
	{
		led_switch(1);				//Change LED to red upon error
		return null_ptr;		//Return Status
	}
	else if ((ptr->cbuffptr)==NULL)
	{
		return buffer_NA;   //Return Status
	}
	else if ((ptr->count)==0)
	{
		led_switch(1);				//Change LED to red upon error
```

```
		return cbuff_empty;	//Return Status
	}
}

/**************************************************************************************************
 * Function Name:cbuff_status cbuff_add(cbuff *ptr, uint8_t val)
 * Description :This Function adds the value to the address pointed by the head of the circular buffer
 * @input: pointer to circular buffer and value to be added
 * @Return : error status messages
**************************************************************************************************/

cbuff_status cbuff_add(cbuff *ptr, uint8_t val)
{
	if(ptr==NULL)
	{
		led_switch(1);		//Change LED to red upon error
		return null_ptr;	//Return Status
	}
	else if ((ptr->cbuffptr)==NULL)
	{
		return buffer_NA;	//Return Status
	}
	else if (cbuff_check_full(ptr)==cbuff_full)
	{
		led_switch(1);			//Change LED to red upon error
		return cbuff_full;	//Return Status
	}
	else if(ptr->head==((ptr->cbuffptr)+((ptr->size)-1)))	//handle Wrap add Condition
```

```
	{
		*(ptr->head)= val;  //Store the value at the address pointed by the head


		printf("WRAP ADD ::New Item Inserted at position %d location :: %d \r
\n",ptr->head,*ptr->head);
		ptr->head=ptr->cbuffptr;  //Initialize the head to base address of the
cbuff pointer
		ptr->count++;//Increment count
		return wrap_add;    //Return Status
	}
	else
	{
		*(ptr->head)=val;//Store the value at the address pointed by the head
		printf("New Item Inserted at position %d location :: %d  \r \n",ptr-
>head,*ptr->head);
		ptr->head++; //Increment the head pointer
		ptr->count++;//Increment the count
		return cbuff_success;     //Return Status
	}
}


/**************************************************************************
******************
 * Function Name:cbuff_status cbuff_delete(cbuff *ptr, uint8_t *val)
 * Description :This Function deletes  the value to the address pointed by the tail
of the circular buffer
 * @input: pointer to circular buffer and pointer to a location where the removed
value to be stored
 * @Return : error status messages

**************************************************************************
*****************/
cbuff_status cbuff_delete(cbuff *ptr, uint8_t *val)
{
```

```
	if(ptr==NULL)
	{
		led_switch(1);			//Change LED to red upon error
		return null_ptr;	//Return Status
	}
	else if ((ptr->cbuffptr)==NULL)
	{
		led_switch(1);			//Change LED to red upon error
		return buffer_NA;	//Return Status
	}
	else if (cbuff_isempty(ptr)==cbuff_empty)
	{
		led_switch(1);			//Change LED to red upon error
		return cbuff_empty; //Return Status
	}
	else if(ptr->tail==((ptr->cbuffptr)+((ptr->size)-1)))	//Handle Wrap
remove condition
	{
		*(val)=*(ptr->tail);	//Store the value pointed by the tail to a
memory address
		ptr->tail=ptr->cbuffptr;  //point tail to the base address of the
circular buffer
		ptr->count--;//Decrement count
		if(a==0 || a==1)
			Log_String(a,cbuffdelete,"Wrap- Deleted");	//T

		return wrap_remove; //Return Status
	}
	else
	{
		*(val)=*(ptr->tail);	//Store the value pointed by the tail to a
memory address
		ptr->tail++; //Increment the tail address
```

```
		ptr->count--;//Decrement count
		if(a==0 || a==1)
			Log_String(a,cbuffdelete,"Deleted");	//T

		return cbuff_success;	//Return status
	}

}
/*************************************************************************************************
 * Function Name:cbuff_status verify_init(cbuff* ptr)
 * Description :This Function verifies whether a circular buffer is initialized or not by checking the pointer
 * @input: pointer to circular buffer.
 * @Return : error status messages

*************************************************************************************************/
cbuff_status verify_init(cbuff* ptr)
{
	if(ptr->cbuffptr==NULL)
	{
		led_switch(1);		//Change LED to red upon error
		return buffer_init_failed;//Return status
	}
	else
	{
		return buffer_init_success;	//Return Status
	}
}

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

```
 * Function Name:cbuff_status verify_ptr(cbuff *ptr)

 * Description :This Function verifies whether a pointer is within the range of
circular buffer

 * @input: pointer to circular buffer.

 * @Return : error status messages

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


cbuff_status verify_ptr(uint8_t *ptr1,cbuff *ptr)

{

      if(ptr1 >= ptr->cbuffptr && ptr1 <= ptr->head )     //Check whether passed
pointer is in the range of circular buffer

      {


            return ptr_valid;   //return status

      }

      else

      {

            led_switch(1);                  //Change LED to red upon error

            return ptr_invalid;       //Return Status

      }


}


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

 * Function Name:cbuff_status cbuff_resize(cbuff *ptr,uint8_t length)

 * Description :This Function resizes the circular buffer to the size of length
passed as parameter implementing realloc function

 * @input: pointer to circular buffer and new length.

 * @Return : error status messages
```

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


cbuff_status cbuff_resize(cbuff *ptr,uint8_t length)

{

	if(ptr==NULL)

	{

		led_switch(1);	//Change LED to red upon error

		return null_ptr;	//Return Status

	}

	else if ((ptr->cbuffptr)==NULL)

	{

		return buffer_NA;	//Return Status

	}

	else

	{

		if(a==0 || a==1)

			Log_String(a,cbuffresize,"*** EXTRA CREDIT: BUFFER RESIZED***");
	//T

		ptr->newcbuffptr=(uint8_t *)realloc(ptr-
>cbuffptr,sizeof(uint8_t)*length);	//Reallocate the memory

		ptr->cbuffptr=ptr->newcbuffptr;  //Set pointer value to new memory
location pointed by newcbuff pointer

		//ptr->head =ptr->newcbuffptr;

		ptr->size=length;	//Set size to length passed

		ptr->count =0;		//Rest count to zero

		return cbuff_success;	//Return Status

	}

}


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

 * Function Name:cbuff_status cbuff_resize(cbuff *ptr,uint8_t length)
```

```
 * Description :This Function prints the elements in the circular buffer along with
its location

 * @input: pointer to circular buffer.

 * @Return : void

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


void cbuff_print(cbuff* ptr)

{

	uint8_t *temp = ptr->tail;//Temporary pointer to store address of Out
location(Tail) of the circular buffer

	for (int i=0;i<ptr->count;i++)

	{

		printf(" \r \n value at position %x location :: %d ",temp,*temp);
	//Print the elements of circular buffer

		temp++;	//Increment pointer

	}

}

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

 * Function Name:cbuff_status cbuff_destroy(cbuff* ptr)

 * Description :This Function destroys the memory allocated for circular buffer

 * @input: pointer to circular buffer and new length.

 * @Return : error status messages

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



cbuff_status cbuff_destroy(cbuff* ptr)

{

	if(ptr->cbuffptr==NULL)

	{
```

```
		led_switch(1);			//Change LED to red upon error
		return destroy_failed;		//Return Status
	}
	else
	{
		free(ptr->cbuffptr);		//Free memory allocated to cbuff pointer

		return destroy_pass;		//Return Status
	}
}
```

```
#ifndef LOGGER_H_
#define LOGGER_H_

///////////////ENUM for test, debug and status//////////////////
typedef enum
{
	Test,
	Debug,
	Normal
}modes ;

////////////////////////////////////////////////////////////

typedef enum
{
	cbuffinit,
	cbuffcheck_full,
	cbuffisempty,
	cbuffadd,
	cbuffdelete,
	verifyinit,
	verifyptr,
	cbuffresize,
	cbuffprint,
	InitUART0,
	Uartrx,
	Uarttx,
	Transmitwait,
	Recievewait,
	UART0printstring,
	UART0print_int,
	putchcbuff,
	UART0IRQHandler,
	cbuffstring,
	Getinfo,
	charactercount,
```

```
	Application_poll,
	Application_int,
	Echo_function_poll,
	Echo_function_interrupt,
	Generate_report,
	Main

}fnnames;

void Log_enable();
void Log_disable();
uint8_t Log_status();
void Log_data (uint32_t *, uint32_t );
void Log_String(uint8_t ,uint8_t,char *str);
void Log_integer(uint8_t ,int16_t);
uint8_t Log_level();

#endif



/*
 * logger.c
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE & Atharv Desai
 */
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "logger.h"
#include "main.h"
#include "uart_interrrupt.h"
#include "circularbuff.h"
#include "time_stamp.h"

modes a = Debug;   // setting mode

fnnames fn_name;

////////TABLE for modes and current states////////////////////////////////

uint8_t flag;
////// Logger for integer ///////////

/*****************************************************************************
***************
 * Function Name: Log_integer(uint32_t intval)
 * Description : This function prints the integer value to the serial terminal
 * @input: integer to be printed
 * @Return : void

*****************************************************************************
**************/
void Log_integer(modes current_mode,int16_t intval)
{
	if(current_mode != 1)	// since no integers to print in normal status mode
	//printf("%d ",intval); // Print the  data
```

```
	UART0_print_int(intval);
}


/*****************************************************************************************
**************
 * Function Name: Log_string(char* str)
 * Description : This function prints the string pointed by the input argument
 * @input:  pointer from which string  to be printed
 * @Return : void

*****************************************************************************************
**************/
////// Logger for string ///////////
void Log_String(uint8_t current_mode,fnnames mycurrent_function, char *str)
{
	time_stamp_print();		//Print time stamp
	if (current_mode ==0)
	{
	    char *s ="\t Test Mode: ";
	    UART0_print_string(s);
	}
	 if (current_mode ==1)
	{
		char *s =" \t Debug Mode: ";
		UART0_print_string(s);
	}
	 if (current_mode ==2)
	{
			char *s ="\t Normal Mode: ";
			UART0_print_string(s);
	}


	 UART0_print_string(str);


	if (mycurrent_function==cbuffinit)
	{
		UART0_print_string("\t Function: cbuffinit \n");
	}
	else if (mycurrent_function==cbuffcheck_full)
	{
		UART0_print_string("\t Function: cbuffcheck_full \n");
	}
	else if (mycurrent_function==cbuffisempty)
	{
		UART0_print_string("\t Function: cbuffisempty \n");
	}
	else if (mycurrent_function==cbuffadd)
	{
		UART0_print_string("\t Function: cbuffadd \n");
	}
	else if (mycurrent_function==cbuffdelete)
	{
		UART0_print_string("\t Function: cbuffdelete \n");
	}
	else if (mycurrent_function==verifyinit)
```

```
{
	UART0_print_string("\t Function: verifyinit \n");
}
else if (mycurrent_function==verifyptr)
{
	UART0_print_string("\t Function: verifyptr \n");
}
else if (mycurrent_function==cbuffresize)
{
	UART0_print_string("\t Function: cbuffresize \n");
}
else if (mycurrent_function==cbuffprint)
{
	UART0_print_string("\t Function: cbuffprint \n");
}
else if (mycurrent_function==InitUART0)
{
	UART0_print_string("\t Function: InitUART0 \n");
}
else if (mycurrent_function==Uartrx)
{
	UART0_print_string("\t Function: Uartrx \n");
}
else if (mycurrent_function==Uarttx)
{
	UART0_print_string("\t Function:   Uarttx \n");
}
else if (mycurrent_function==Transmitwait)
{
	UART0_print_string("\t Function:   Transmitwait \n");
}
else if (mycurrent_function==Recievewait)
{
	UART0_print_string("\t Function:Recievewait \n");
}
else if (mycurrent_function==UART0printstring)
{
	UART0_print_string("\t Function: UART0printstring \n");
}
else if (mycurrent_function==UART0print_int)
{
	UART0_print_string("\t Function: UART0print_int \n");
}
else if (mycurrent_function==putchcbuff)
{
	UART0_print_string("\t Function: putchcbuff \n");
}
else if (mycurrent_function==UART0IRQHandler)
{
	UART0_print_string("\t Function: UART0IRQHandler \n");
}
else if (mycurrent_function==cbuffstring)
{
	UART0_print_string("\t Function: cbuffstring \n");
}
else if (mycurrent_function==Getinfo)
{
	UART0_print_string("\t Function: Getinfo \n");
```

```
	}
	else if(mycurrent_function==charactercount)
	{
		UART0_print_string("\t Function: charactercount \n");
	}
	else if(mycurrent_function==Application_poll)
		{
			UART0_print_string("\t Function: Application_poll \n");
		}
	else if(mycurrent_function==Application_int)
		{
			UART0_print_string("\t Function: Application_int \n");
		}
	else if(mycurrent_function==Echo_function_poll)
			{
				UART0_print_string("\t Function: echo_function_poll \n");
			}
	else if(mycurrent_function==Echo_function_interrupt)
			{
				UART0_print_string("\t Function: echo_function_interrupt \n");
			}
	else if(mycurrent_function==Generate_report)
			{
				UART0_print_string("\t Function: generate_report \n");
			}
	else if(mycurrent_function==Main)
			{
				UART0_print_string("\t Function: Main \n");
			}

}

///////////////Log level//////////////////////////
uint8_t Log_level()
{

	return a;
}


/*
 * Copyright 2016-2019 NXP
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without modification,
 * are permitted provided that the following conditions are met:
 *
 * o Redistributions of source code must retain the above copyright notice, this list
 *   of conditions and the following disclaimer.
 *
 * o Redistributions in binary form must reproduce the above copyright notice, this
 *   list of conditions and the following disclaimer in the documentation and/or
 *   other materials provided with the distribution.
 *
 * o Neither the name of NXP Semiconductor, Inc. nor the names of its
 *   contributors may be used to endorse or promote products derived from this
 *   software without specific prior written permission.
 *
```

```
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE FOR
 * ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
 * (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
 * ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
 * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

/**
 * @file    mtb.c
 * @brief   MTB initialization file.
 * @details Symbols controlling behavior of this code...
 *                  __MTB_DISABLE
 *                  If this symbol is defined, then the buffer array for the MTB
 *                  will not be created.
 *
 *                  __MTB_BUFFER_SIZE
 *                  Symbol specifying the sizer of the buffer array for the MTB.
 *                  This must be a power of 2 in size, and fit into the available
 *            RAM. The MTB buffer will also be aligned to its 'size'
 *                  boundary and be placed at the start of a RAM bank (which
 *                  should ensure minimal or zero padding due to alignment).
 *
 *                  __MTB_RAM_BANK
 *                  Allows MTB Buffer to be placed into specific RAM bank. When
 *                  this is not defined, the "default" (first if there are
 *                  several) RAM bank is used.
 */

/* This is a template for board specific configuration created by MCUXpresso IDE Project
Wizard.*/

// Allow MTB to be removed by setting a define (via command line)
#if !defined (__MTB_DISABLE)

  // Allow for MTB buffer size being set by define set via command line
  // Otherwise provide small default buffer
  #if !defined (__MTB_BUFFER_SIZE)
    #define __MTB_BUFFER_SIZE 128
  #endif

  // Check that buffer size requested is >0 bytes in size
  #if (__MTB_BUFFER_SIZE > 0)
    // Pull in MTB related macros
    #include <cr_mtb_buffer.h>

    // Check if MYTB buffer is to be placed in specific RAM bank
    #if defined(__MTB_RAM_BANK)
           // Place MTB buffer into explicit bank of RAM
           __CR_MTB_BUFFER_EXT(__MTB_BUFFER_SIZE,__MTB_RAM_BANK);
    #else
           // Place MTB buffer into 'default' bank of RAM
           __CR_MTB_BUFFER(__MTB_BUFFER_SIZE);
```

```
    #endif  // defined(__MTB_RAM_BANK)

  #endif // (__MTB_BUFFER_SIZE > 0)

#endif // !defined (__MTB_DISABLE)
```

```
/*
 * RGBled.h
 *
 *  Created on: Sep 28, 2019
 *   Author:SURAJ THITE , ATHARV DESAI
 */

#ifndef RGBLED_H_
#define RGBLED_H_
void led_switch(int n);     //Function to switch the led_state
void RGB_init();     //Function to initialize the RGB Leds
void RGB_OFF();      //Function to turn off the RGB led off
void delay(int time_ms); // Delay
#endif /* RGBLED_H_ */
```

```
/*
 * RGBled.c
 *
 *  Created on: Sep 28, 2019
 *      Author:SURAJ THITE , ATHARV DESAI
 */



#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_gpio.h"

#include "clock_config.h"
#include "pin_mux.h"

/*******************************************************************************/
/* Function name:RGB_init
 * Parameters: void
 * Return : void
 * Description: Function to initialize the GPIO RGB Led Pins . */
/*******************************************************************************/
void RGB_init()
{
              gpio_pin_config_t led_blue_config = {
               kGPIO_DigitalOutput, 1,
           }; //Config the pin  for BLUE LED to Digital Output
              GPIO_PinInit(BOARD_LED_BLUE_GPIO,BOARD_LED_BLUE_GPIO_PIN,
&led_blue_config);
           gpio_pin_config_t led_red_config = {
               kGPIO_DigitalOutput, 1,
           }; //Config the pin  for RED LED to Digital Output
           GPIO_PinInit(BOARD_LED_RED_GPIO,BOARD_LED_RED_GPIO_PIN, &led_red_config);
```

```
            gpio_pin_config_t led_green_config = {
                kGPIO_DigitalOutput, 1,
            }; //Config the pin  for GREEN LED to Digital Output
            GPIO_PinInit(BOARD_LED_GREEN_GPIO,BOARD_LED_GREEN_GPIO_PIN,
&led_green_config);  //Initialize the GPIO Pins
}


/*************************************************************************/
/* Function name:led_switch(int n )
 * Parameters: current state n
 * Return : void
 * Description: Function to initialize the GPIO RGB Led Pins . */
/*************************************************************************/
void led_switch(int n)
 {
       GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
       //Clear the Pins
       GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
       GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);

       switch (n)
       {
       // Switch LED BLUE ON and TURN OTHER LEDs OFF
       case 0:
              {
              GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
              GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
              GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
              delay(100);
              }
              break;
              // Switch LED RED ON and TURN OTHER LEDs OFF
       case 1:
       {

                     GPIO_ClearPinsOutput(BOARD_LED_RED_GPIO, 1u <<
BOARD_LED_RED_GPIO_PIN);
                     GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u <<
BOARD_LED_BLUE_GPIO_PIN);
                     GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u <<
BOARD_LED_GREEN_GPIO_PIN);
                     delay(100);
       }
                     break;
                     // Switch LED GREEN ON and TURN OTHER LEDs OFF
       case 2:
       {
                     GPIO_ClearPinsOutput(BOARD_LED_GREEN_GPIO, 1u <<
BOARD_LED_GREEN_GPIO_PIN);
                     GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u <<
BOARD_LED_RED_GPIO_PIN);
                     GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u <<
BOARD_LED_BLUE_GPIO_PIN);
                     //delay(1000);
       }
                     break;
       case 3:
```

```
		{
			// Switch LED BLUE ON and TURN OTHER LEDs OFF
		GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
		GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
		GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
		}
		break;
	}
 }


/*!*****************************************************************************************
**************
 * Function Name:int delay(int time_ms)
 * Description : this function provides delay in milliseconds according to input
parameters
 * @input:time in milliseconds
 * @Return : NULL

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

void delay(int time_ms)
{
	volatile uint32_t i = 0;
	for (i = 0; i < 2400*time_ms; ++i)
	{
		__asm("NOP"); /* No operation */
	}
}

/*!*************************************************************************/
/* Function name:RGB_off
 * Parameters: void
 * Return : void
 * Description: Function to turn off the RGB Led Pins . */
/*!*************************************************************************/

void RGB_OFF()
{	// Clear all the  LEDs.
	GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1u << BOARD_LED_BLUE_GPIO_PIN);
	GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1u << BOARD_LED_GREEN_GPIO_PIN);
	GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1u << BOARD_LED_RED_GPIO_PIN);
}



// ***************************************************************************
// semihost_hardfault.c
//                - Provides hard fault handler to allow semihosting code not
//                  to hang application when debugger not connected.
//
// ***************************************************************************
// Copyright 2017-2019 NXP
// All rights reserved.
//
// Software that is described herein is for illustrative purposes only
// which provides customers with programming information regarding the
```

```
// NXP Cortex-M based MCUs. This software is supplied "AS IS" without any
// warranties of any kind, and NXP Semiconductors and its licensor disclaim any
// and all warranties, express or implied, including all implied warranties of
// merchantability, fitness for a particular purpose and non-infringement of
// intellectual property rights.  NXP Semiconductors assumes no responsibility
// or liability for the use of the software, conveys no license or rights under
// any patent, copyright, mask work right, or any other intellectual property
// rights in or to any products. NXP Semiconductors reserves the right to make
// changes in the software without notification. NXP Semiconductors also makes
// no representation or warranty that such application will be suitable for the
// specified use without further testing or modification.
//
// Permission to use, copy, modify, and distribute this software and its
// documentation is hereby granted, under NXP Semiconductors' and its
// licensor's relevant copyrights in the software, without fee, provided that it
// is used in conjunction with NXP Semiconductors microcontrollers.  This
// copyright, permission, and disclaimer notice must appear in all copies of
// this code.
// ****************************************************************************
//
//                        ===== DESCRIPTION =====
//
// One of the issues with applications that make use of semihosting operations
// (such as printf calls) is that the code will not execute correctly when the
// debugger is not connected. Generally this will show up with the application
// appearing to just hang. This may include the application running from reset
// or powering up the board (with the application already in FLASH), and also
// as the application failing to continue to execute after a debug session is
// terminated.
//
// The problem here is that the "bottom layer" of the semihosted variants of
// the C library, semihosting is implemented by a "BKPT 0xAB" instruction.
// When the debug tools are not connected, this instruction triggers a hard
// fault - and the default hard fault handler within an application will
// typically just contains an infinite loop - causing the application to
// appear to have hang when no debugger is connected.
//
// The below code provides an example hard fault handler which instead looks
// to see what the instruction that caused the hard fault was - and if it
// was a "BKPT 0xAB", then it instead returns back to the user application.
//
// In most cases this will allow applications containing semihosting
// operations to execute (to some degree) when the debugger is not connected.
//
// == NOTE ==
//
// Correct execution of the application containing semihosted operations
// which are vectored onto this hard fault handler cannot be guaranteed. This
// is because the handler may not return data or return codes that the higher
// level C library code or application code expects. This hard fault handler
// is meant as a development aid, and it is not recommended to leave
// semihosted code in a production build of your application!
//
// ****************************************************************************

// Allow handler to be removed by setting a define (via command line)
#if !defined (__SEMIHOST_HARDFAULT_DISABLE)
```

```
__attribute__((naked))
void HardFault_Handler(void){
    __asm(  ".syntax unified\n"
        // Check which stack is in use
            "MOVS   R0, #4  \n"
            "MOV    R1, LR  \n"
            "TST    R0, R1  \n"
            "BEQ    _MSP    \n"
            "MRS    R0, PSP \n"
            "B  _process      \n"
            "_MSP:  \n"
            "MRS    R0, MSP \n"
        // Load the instruction that triggered hard fault
        "_process:     \n"
            "LDR    R1,[R0,#24] \n"
            "LDRH    R2,[r1] \n"
        // Semihosting instruction is "BKPT 0xAB" (0xBEAB)
            "LDR    R3,=0xBEAB \n"
            "CMP     R2,R3 \n"
            "BEQ    _semihost_return \n"
        // Wasn't semihosting instruction so enter infinite loop
            "B . \n"
        // Was semihosting instruction, so adjust location to
        // return to by 1 instruction (2 bytes), then exit function
        "_semihost_return: \n"
            "ADDS    R1,#2 \n"
            "STR    R1,[R0,#24] \n"
       // Set a return value from semihosting operation.
       // 32 is slightly arbitrary, but appears to allow most
       // C Library IO functions sitting on top of semihosting to
       // continue to operate to some degree
                  "MOVS   R1,#32 \n"
                  "STR R1,[ R0,#0 ] \n" // R0 is at location 0 on stack
       // Return from hard fault handler to application
            "BX LR \n"
        ".syntax divided\n") ;
}

#endif


/*****************************************************************************
 *                                                                           *
 *  uCUnit - A unit testing framework for microcontrollers                   *
 *                                                                           *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                       *
 *                  https://www.ucunit.org                                   *
 *                                                                           *
 *  File        : System.h                                                   *
 *  Description : System dependent functions used by uCUnit.                 *
 *  Author      : Sven Stefan Krauss                                         *
 *  Contact     : www.ucunit.org                                             *
 *                                                                           *
 *****************************************************************************/

/*
 * This file is part of ucUnit.
 *
```

```
 * You can redistribute and/or modify it under the terms of the
 * Common Public License as published by IBM Corporation; either
 * version 1.0 of the License, or (at your option) any later version.
 *
 * uCUnit is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * Common Public License for more details.
 *
 * You should have received a copy of the Common Public License
 * along with uCUnit.
 *
 * It may also be available at the following URL:
 *       http://www.opensource.org/licenses/cpl1.0.txt
 *
 * If you cannot obtain a copy of the License, please contact the
 * author.
 */
#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 * msg);
void System_WriteInt(int n);

#endif /* SYSTEM_H_ */


/****************************************************************************
 *                                                                          *
 *  uCUnit - A unit testing framework for microcontrollers                  *
 *                                                                          *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                      *
 *                  https://www.ucunit.org                                  *
 *                                                                          *
 *  File        : System.c                                                  *
 *  Description : System dependent functions used by uCUnit.                *
 *                This file runs with arm-elf-run                           *
 *  Author      : Sven Stefan Krauss                                        *
 *  Contact     : www.ucunit.org                                            *
 *                                                                          *
 ****************************************************************************/

/*
 * This file is part of ucUnit.
 *
 * You can redistribute and/or modify it under the terms of the
 * Common Public License as published by IBM Corporation; either
 * version 1.0 of the License, or (at your option) any later version.
 *
 * uCUnit is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * Common Public License for more details.
```

```
 *
 * You should have received a copy of the Common Public License
 * along with uCUnit.
 *
 * It may also be available at the following URL:
 *       http://www.opensource.org/licenses/cpl1.0.txt
 *
 * If you cannot obtain a copy of the License, please contact the
 * author.
 */
#include <stdio.h>
#include <stdlib.h>
#include "System.h"

/* Stub: Initialize your hardware here */
void System_Init(void)
{

       printf("Init of hardware finished.\n");
}

/* Stub: Shutdown your hardware here */
void System_Shutdown(void)
{

       /* asm("\tSTOP"); */
       printf("System shutdown.\n");
       exit(0);
}

/* Stub: Recover the system */
void System_Recover(void)
{
       /* Stub: Recover the hardware */
       /* asm("\tRESET"); */
       printf("System reset.\n");
       exit(0);
}

/* Stub: Put system in a safe state */
void System_Safestate(void)
{
       /* Disable all port pins */
       /* PORTA = 0x0000; */
       /* PORTB = 0x0000; */
       /* PORTC = 0x0000; */

       /* Disable interrupts */
       /* DIE(); */

       /* Put processor into idle state */
       /* asm("\tIDLE"); */
       printf("System safe state.\n");
       exit(0);
}

/* Stub: Transmit a string to the host/debugger/simulator */
void System_WriteString(char * msg)
```

```
{
	printf(msg);
}

void System_WriteInt(int n)
{
	printf("%i", n);
}


/*
 * time_stamp.h
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE & ATHARV DESAI
 */

#ifndef TIME_STAMP_H_
#define TIME_STAMP_H_

#include "uart_interrrupt.h"
#include "circularbuff.h"

void Init_SysTick(void);
void SysTick_Handler();
uint64_t get_current_time();
uint64_t time_passed(uint64_t since);
void time_stamp_print();

#endif /* TIME_STAMP_H_ */


/*
 * timer_stamp.c
 *
 *  Created on: Nov 17, 2019
 *      Author: SURAJ THITE ,Atrharv Desai
 */

#include "time_stamp.h"
#include "main.h"
#include "MKL25Z4.h"
static uint64_t current_time = 0;
static const uint64_t time_max = ~0;
/**************************************************************************************
***************
 * Function Name:Init_SysTick(void)
 * Description :This function Initializes the SysTick Timer for 0.1 second interrupt.
 * @input: void
 * @Return : Void

**************************************************************************************
**************/
void Init_SysTick(void)
{
	SysTick->LOAD = (48000000L/100);   //Initialize Load value
	NVIC_SetPriority(SysTick_IRQn,3);        //Enable NVIC Interrupt with priority 3
	NVIC_EnableIRQ(SysTick_IRQn);      //Enable NVIC IRQ
```

```
	SysTick->VAL=0;	//Set VAL =0
	SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;  //Enable
interrupt
}

/**************************************************************************************
**************
 * Function Name:void SysTick_Handler()
 * Description :This function is the IRQ Handler which increments global varibale value
for current time in tenths of seconds
 * @input: void
 * @Return : void

**************************************************************************************
**************/
//Event handler for SystickTimer for 15 seconds delay
void SysTick_Handler()
{
	current_time++;	//Increment the gloabl variable
}

/**************************************************************************************
**************
 * Function Name:uint64_t get_current_time()
 * Description :This function returns current Systick Counter ValueE
 * @input: void
 * @Return : void

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

uint64_t get_current_time()
{
	return current_time; //Return current time
}

/**************************************************************************************
**************
 * Function Name:uint64_t time_passed(uint64_t since)
 * Description :This function returns time elapsed since the bootup
 * Reference from "Making Embedded Systems: Design Patterns for Great Software ;Elecia
White Book"
 * @input: void
 * @Return : void

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

uint64_t time_passed(uint64_t since)
{
	uint64_t now = current_time;

	if(now >= since)
	{
		return now - since;
	}

	return (now + (time_max-since));
```

```
}

/*****************************************************************************************
***************
 * Function Name:void time_stamp_print()
 * Description :This function prints the time stamp on host connected to the UART0
terminal
 * @input: void
 * @Return : void

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

void time_stamp_print()
{
	static char time_buf[2048] = {0};
	for(int i = 0; i < 2048; i++) time_buf[i] = '\0';	//Initialize array with
nullCharacters

	uint64_t tenths_count = get_current_time();	//Get current time

	float current = tenths_count / 10;

	//Calculations for conversion to Hours , mins , seconds
	uint64_t sec = (uint64_t)(current)%60;
	uint64_t min = (uint64_t)(current/60)%60;
	uint64_t hrs = (uint64_t)(current/3600)%60;


	sprintf(time_buf, "\n%02d:",  hrs);	//Convert hrs to string
	UART0_print_string(time_buf);	//Send value over UART
	sprintf(time_buf, "%02d:",  min);//Convert min to string
	UART0_print_string(time_buf);//Send value over UART
	sprintf(time_buf, "%02d:",  sec);//Convert sec to string
	UART0_print_string(time_buf);//Send value over UART
	sprintf(time_buf, ".%1d",  tenths_count%10);//Convert tenths_count to string
	UART0_print_string(time_buf);//Send value over UART
}


/*
 * uart.h
 *
 *  Created on: Nov 9, 2019
 *      Author: SURAJ THITE & ATHARV DESAI
 */

#ifndef UART_INTERRRUPT_H_
#define UART_INTERRRUPT_H_

#include "main.h"
#include "MKL25Z4.h"
#include "circularbuff.h"
#include "fsl_debug_console.h"
//void UART_configure(void);
```

```
#define USE_UART_INTERRUPTS       (1) // 0 for polled UART communications, 1 for
interrupt-driven
#define UART_OVERSAMPLE_RATE      (16)
#define BUS_CLOCK                       (24e6)
#define SYS_CLOCK                       (48e6)
#define  START_CRITICAL()  __disable_irq()
#define  END_CRITICAL()   __enable_irq()
void tx_poll();
void custom_printf(char *);
void Init_UART0(uint32_t baud_rate);
void transmit_wait();
void recieve_wait();
void UART0_print_string(char *str);
char uart_rx(void);
void uart_tx(char ch);
void UART0_print_int(uint16_t count);
#endif /* UART_INTERRRUPT_H_ */


/*
 * uart.c
 *
 *  Created on: Nov 9, 2019
 *  Author: SURAJ THITE ,Atharv Desai
 *  Reference for Initializing Logger : https://github.com/alexander-g-
dean/ESF/blob/master/Code/Chapter_8/Serial-Demo/inc/UART.h
 */

#include "uart_interrrupt.h"
#include "logger.h"
#include "RGBled.h"

//Global Variables Access//
char ch1;
uint8_t deleted_element;
extern cbuff *rx;
extern uint8_t *element_deleted;
extern uint8_t info[256];
extern int tx_flag;
extern bool rx_flag;
extern bool rx_flag_1;
extern modes a;
int wait_flag;
int8_t rx_data;



/*****************************************************************************
***************
 * Function Name:Init_UART0(uint32_t baud_rate)
 * Description :This function initializes the UART0 with selected baud rate as input
 * @input: BAUD
 * @Return : void
 * Reference :https://github.com/alexander-g-dean/ESF/blob/master/Code/Chapter_8/Serial-
Demo/inc/UART.h
```

```
*****************************************************************************
**************/
void Init_UART0(uint32_t baud_rate)
{
	uint16_t sbr;
	uint8_t temp;

	// Enable clock gating for UART0 and Port A
	SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
	SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;

	// Make sure transmitter and receiver are disabled before init
	UART0->C2 &= ~UART0_C2_TE_MASK & ~UART0_C2_RE_MASK;

	// Set UART clock to 48 MHz clock
	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

	// Set baud rate and oversampling ratio
	sbr = (uint16_t)((SYS_CLOCK)/(baud_rate * UART_OVERSAMPLE_RATE));
	UART0->BDH &= ~UART0_BDH_SBR_MASK;
	UART0->BDH |= UART0_BDH_SBR(sbr>>8);
	UART0->BDL = UART0_BDL_SBR(sbr);
	UART0->C4 |= UART0_C4_OSR(UART_OVERSAMPLE_RATE-1);

	// Disable interrupts for RX active edge and LIN break detect, select one stop bit
	UART0->BDH |= UART0_BDH_RXEDGIE(0) | UART0_BDH_SBNS(0) | UART0_BDH_LBKDIE(0);

	// Don't enable loopback mode, use 8 data bit mode, don't use parity
	UART0->C1 = UART0_C1_LOOPS(0) | UART0_C1_M(0) | UART0_C1_PE(0);
	// Don't invert transmit data, don't enable interrupts for errors
	UART0->C3 = UART0_C3_TXINV(0) | UART0_C3_ORIE(0)| UART0_C3_NEIE(0)

	| UART0_C3_FEIE(0) | UART0_C3_PEIE(0);

	// Clear error flags
	UART0->S1 = UART0_S1_OR(1) | UART0_S1_NF(1) | UART0_S1_FE(1) | UART0_S1_PF(1);

	// Try it a different way
	UART0->S1 |= UART0_S1_OR_MASK | UART0_S1_NF_MASK |
	             UART0_S1_FE_MASK | UART0_S1_PF_MASK;

	// Send LSB first, do not invert received data
	UART0->S2 = UART0_S2_MSBF(0) | UART0_S2_RXINV(0);

#if USE_UART_INTERRUPTS
	// Enable interrupts. Listing 8.11 on p. 234
	//     Q_Init(&TxQ);
	//     Q_Init(&RxQ);

	NVIC_SetPriority(UART0_IRQn, 2); // 0, 1, 2, or 3
	NVIC_ClearPendingIRQ(UART0_IRQn);
	NVIC_EnableIRQ(UART0_IRQn);
```

```
	// Enable receive interrupts but not transmit interrupts yet
	UART0->C2 |= UART_C2_RIE(1);
#endif

	// Enable UART receiver and transmitter
	UART0->C2 |= UART0_C2_RE(1) | UART0_C2_TE(1);

	// Clear the UART RDRF flag
	temp = UART0->D;
	UART0->S1 &= ~UART0_S1_RDRF_MASK;

}
#if USE_UART_INTERRUPTS

/*********************************************************************************
**************
 * Function Name:char uart_rx(void)
 * Description :This Non-Blocking function to receive the character over the UART0 and
return
 * @input: void
 * @Return : char

*********************************************************************************
**************/
char uart_rx(void)
{
	return UART0->D;	//Return value recieved at the recieve buffer
}

/*********************************************************************************
**************
 * Function Name:void uart_tx(char ch)
 * Description :Non-Blocking function to send the character over the UART0
 * @input: char
 * @Return : void

*********************************************************************************
**************/
void uart_tx(char ch)
{
	UART0->D = ch;	//Store the character value in the transmit buffer
}
#else
/*********************************************************************************
**************
 * Function Name:char uart_rx(void)
 * Description :Blocking function to receive the character over the UART0
 * @input: void
 * @Return : char

*********************************************************************************
**************/
char uart_rx(void)
{
	recieve_wait();	//Wait for character to recieve
	rx_flag_1 =1; //Set the Flag
	return UART0->D;	//Return the received data
```

```
}

/**********************************************************************************
**************
 * Function Name:char uart_rx(void)
 * Description :Blocking function to send the character over the UART0
 * @input: char
 * @Return : void

**********************************************************************************
**************/
void uart_tx(char ch)
{
	transmit_wait();	//Wait for Tx interrupt
	UART0->D = ch;		//Store the character value in transmit buffer
}
#endif

/**********************************************************************************
**************
 * Function Name:void transmit_wait()
 * Description :This function waits for Tx flag
 * @input: void
 * @Return : void

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


void transmit_wait()
{
	led_switch(2);	//Set LED to green
	while (!(UART0->S1 & UART_S1_TDRE_MASK));//Wait for Tx
}

/**********************************************************************************
**************
 * Function Name:char uart_rx(void)
 * Description :This function waits for Tx flag
 * @input: void
 * @Return : void

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

void recieve_wait()
{
	led_switch(0);	//Switch led to Blue
	if (a==1)
		Log_String(a,Recievewait,"Waiting for Character to receive");
	while(!(UART0->S1 & UART_S1_RDRF_MASK));  //Wait for Rx
}

/**********************************************************************************
**************
 * Function Name:void UART0_print_string(char *str)
 * Description :This function Prints the string on by transmitting it through the UART
terminal
```

```
 * @input: string
 * @Return : void

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

void UART0_print_string(char *str)
{
	while(*str != '\0')
	{
		transmit_wait();	//Wait for Tx
		uart_tx(*str);		//Transmit character of the string
		str++; //Increment the pointer
	}
}

/*********************************************************************************************
***************
 * Function Name:void UART0_print_int(uint16_t count)
 * Description :This function prints the integer to the uart terminal
 * @input: integer value to be transmitted
 * @Return : void

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


void UART0_print_int(uint16_t count)
{
	char str[10]; //Temporary array
	sprintf(str,"%d",count);	//Convert integer value to char
	UART0_print_string(str);	//Transmit the string via UART
}

/*********************************************************************************************
***************
 * Function Name:void putch_cbuff(char ch)
 * Description :This function adds the value received to the UART terminal
 * @input: char
 * @Return : void

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


void putch_cbuff(char ch)
{
	cbuff_status overflow = cbuff_add(rx,ch); //Check whether circular buffer is
overflowed
	if (overflow == cbuff_full)
	{
		if(a==1 || a==0)
		{
			Log_String(a,putchcbuff,"Buffer_Full"); //T
			Log_String(a,putchcbuff,"Resizing the buffer");
			cbuff_resize(rx,20); //Resize the buffer if overflow status recieved
= cbuff_full
```

```
		}
	}
}

/*****************************************************************************************
***************
 * Function Name:void UART0_IRQHandler()
 * Description :This is the interrupt handler for UART0
 * @input: void
 * @Return : void

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


void UART0_IRQHandler()
{
	START_CRITICAL();			//START OF CRITICAL REGION
	//Handling of Errors
	if (UART0->S1 & (UART_S1_OR_MASK |UART_S1_NF_MASK | UART_S1_FE_MASK |
UART_S1_PF_MASK))
	{
		//Change LED to RED
		led_switch(1);
		UART0->S1 |= UART0_S1_OR_MASK | UART0_S1_NF_MASK | UART0_S1_FE_MASK |
UART0_S1_PF_MASK;
		ch1 = UART0->D;
	}

	//Interrupt Handler for  transmit interrupt
	if(UART0->S1 & UART0_S1_TDRE_MASK)
	{
		UART0->C2 &= ~(UART0_C2_TIE_MASK);

	}
	//Interrupt Handler for  Rx interrupt
	if(UART0->S1 & UART0_S1_RDRF_MASK)
	{
		led_switch(0);		//Change LED to blue
		ch1=UART0->D; //store received character into variable
		if(a==1)
			Log_String(a,UART0IRQHandler,"RX Interrupt Detected");
		putch_cbuff(ch1); //Store recieved character into circular buffer
		rx_flag = 1; //Set flag to 1
	}
	END_CRITICAL();		//END OF CRITICAL REGION
}

/*****************************************************************************************
***************
 * Function Name:void cbuff_string(char *str)
 * Description :This function adds the string to the circular buffer
 * @input: char
 * @Return : void

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

```
void cbuff_string(char *str)
{

	while(*str != '\0')
	{
		if (a==1)
			Log_String(a,cbuffstring,"Adding String to Circular Buffer");
		putch_cbuff(*str);   //Print character pointed by str
		str++; //Increment the pointer
	}
}
/*****************************************************************************************
***************
 * Function Name:void getinfo(uint8_t *element_deleted)
 * Description :This function updates the count of elements that have been deleted
 * @input: pointer to the deleted element
 * @Return : void

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

void getinfo(uint8_t *element_deleted)
{
	info[*(element_deleted)]++;
}


/*****************************************************************************
 *                                                                           *
 *  uCUnit - A unit testing framework for microcontrollers                   *
 *                                                                           *
 *  (C) 2007 - 2008 Sven Stefan Krauss                                       *
 *                  https://www.ucunit.org                                   *
 *                                                                           *
 *  File        : uCUnit-v1.0.h                                              *
 *  Description : Macros for Unit-Testing                                    *
 *  Author      : Sven Stefan Krauss                                         *
 *  Contact     : www.ucunit.org                                             *
 *                                                                           *
 *****************************************************************************/

/*
 * This file is part of ucUnit.
 *
 * You can redistribute and/or modify it under the terms of the
 * Common Public License as published by IBM Corporation; either
 * version 1.0 of the License, or (at your option) any later version.
 *
 * uCUnit is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * Common Public License for more details.
 *
 * You should have received a copy of the Common Public License
 * along with uCUnit.
 *
 * It may also be available at the following URL:
 *       http://www.opensource.org/licenses/cpl1.0.txt
```

```
 *
 * If you cannot obtain a copy of the License, please contact the
 * author.
 */

#ifndef UCUNIT_0101_H_
#define UCUNIT_0101_H_

/*****************************************************************************/
/****** Customizing area ******/
/*****************************************************************************/

/**
 * @Macro:       UCUNIT_WriteString(msg)
 *
 * @Description: Encapsulates a function which is called for
 *               writing a message string to the host computer.
 *
 * @param msg:   Message 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_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 */
```

```
/*************************************************************************/
/* 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(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(": 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(__FILE__);                           \
        UCUNIT_WriteString(":");                                \
        UCUNIT_WriteString(UCUNIT_DefineToString(__LINE__));    \
        UCUNIT_WriteString(": 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", #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, "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", #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", #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", #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.
```

```
 *
 */
#define UCUNIT_TestcaseBegin(name)                                          \
    do                                                                      \
    {                                                                       \
        UCUNIT_WriteString("\n====================================="); \
        UCUNIT_WriteString(name);                                           \
        UCUNIT_WriteString("====================================="); \
        ucunit_testcases_failed_checks = ucunit_checks_failed;              \
    }                                                                       \
    while(0)

/**
 * @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");  \
        if( 0==(ucunit_testcases_failed_checks - ucunit_checks_failed) ) \
        {                                                                   \
            UCUNIT_WriteString("Testcase passed.\n");                       \
            ucunit_testcases_passed++;                                      \
        }                                                                   \
        else                                                                \
        {                                                                   \
            UCUNIT_WriteFailedMsg("EndTestcase","");                        \
            ucunit_testcases_failed++;                                      \
        }                                                                   \
        UCUNIT_WriteString("=====================================\n"); \
    }                                                                       \
    while(0)

/*****************************************************************************/
/* Support for code coverage */
/*****************************************************************************/

/**
 * @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**************************************");   \
    UCUNIT_WriteString("\nTestcases: failed: ");                    \
    UCUNIT_WriteInt(ucunit_testcases_failed);                       \
    UCUNIT_WriteString("\n           passed: ");                    \
    UCUNIT_WriteInt(ucunit_testcases_passed);                       \
```

```
    UCUNIT_WriteString("\nChecks:    failed: ");                      \
    UCUNIT_WriteInt(ucunit_checks_failed);                            \
    UCUNIT_WriteString("\n           passed: ");                      \
    UCUNIT_WriteInt(ucunit_checks_passed);                            \
    UCUNIT_WriteString("\n**************************************\n"); \
}

#endif /*UCUNIT_H_*/
```

```
/*
 * unitTest.h
 *
 *  Created on: NOV 18, 2019
 *      Author: SURAJ THITE , ATHRRV DESAI
 */

#ifndef UNITTEST_H_
#define UNITTEST_H_
#include "main.h"
#include "System.h"

#include "uCUnit-v1.0.h"
#include "stdint.h"

#include "circularbuff.h"

//Test Suite to run tests
void Testsuite_RunTests(void);//TestSuit


//Test functions

void test_cbuff_init(void);
void test_cbuff_add(void);
void test_cbuff_wrap_add(void);
void test_cbuff_full(void);
void test_delete(void);
void test_head_ptr(void);
void test_cuff_resize(void);
void test_cbuff_destroy(void);

#endif /* UNITTEST_H_ */
```

```
/*
 * unitTest.c
 *
 *  Created on: Oct 22, 2019
 *      Author: SURAJ THITE ,ATHARV DESAI
 */

#include "unitTest.h"
cbuff* test ;

//test-example-1
```

```c
/**********************************************************************************
**************
 * Function:static void test_cbuff_init(void)
 * Description :Test to verify initialization of circular buffer
 * @input:void
 * @Return :void

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

void test_cbuff_init(void)
{
	UCUNIT_TestcaseBegin("TEST1:Initialization of Circular Buffer");
	test= malloc(sizeof(cbuff));
	test->cbuffptr = malloc(sizeof(int8_t) * 10);
	cbuff_status s = cbuff_init(test,10);
	UCUNIT_CheckIsEqual(s,cbuff_init_success); //PASS
	UCUNIT_TestcaseEnd();
}


/**********************************************************************************
**************
 * Function Name:static void test_cbuff_add(void)
 * Description : Test for addition of data into circular buffer
 * @input:void
 * @Return :void

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

//test-example-2
void test_cbuff_add(void)
{
	UCUNIT_TestcaseBegin("TEST2:Test For addition of Elements to circular Buffer");
	cbuff_status s;
	for (int i=0;i<9;i++)
			{
				s = cbuff_add(test,i);
			}
	cbuff_print(test);
	UCUNIT_CheckIsEqual(s,cbuff_success); //PASS
	UCUNIT_TestcaseEnd();
}

/**********************************************************************************
**************
 * Function Name:static void test_cbuff_wrap_add(void)
 * Description : Test for wrap addition
 * @input:void
 * @Return :void

**********************************************************************************
*************/
//test-example-3
void test_cbuff_wrap_add(void)
{
	UCUNIT_TestcaseBegin("TEST3: Test for Wrap Addition");
```

```
	cbuff_status s;
	s = cbuff_add(test,9);
	UCUNIT_CheckIsEqual(s,wrap_add); //PASS
	UCUNIT_TestcaseEnd();
}

/**************************************************************************************
***************
 * Function Name:static void test_cbuf_full(void)
 * Description : This function tests whether circular buffer is full or not
 * @input:void
 * @Return :void

**************************************************************************************
**************/
//test-example-4
void test_cbuff_full(void)
{
	UCUNIT_TestcaseBegin("TEST4: Test for Buffer Full");
	cbuff_status s;
	s = cbuff_add(test,5);
	UCUNIT_CheckIsEqual(s,cbuff_full); //PASS
	UCUNIT_TestcaseEnd();
}
/**************************************************************************************
***************
 * Function Name:static void test_delete(void)
 * Description : Test for checking successful deletion of an element from circular buffer
 * @input:void
 * @Return :void

**************************************************************************************
**************/
//test-example-5
void test_delete(void)
{

	UCUNIT_TestcaseBegin("TEST3:Test Whether deletion was Successful in the Circular
Buffer");
	uint8_t sar =0;
	cbuff_status s = cbuff_delete(test,&sar);
	cbuff_print(test);
	UCUNIT_CheckIsEqual(s,cbuff_success); //PASS
	UCUNIT_TestcaseEnd();

}


/**************************************************************************************
***************
 * Function Name: static void test_head_ptr(void)
 * Description : Test to check validity of circular buffer head pointer
 * @input:void
 * @Return :void.

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

```
//test-example-6
void test_head_ptr(void)
{
	UCUNIT_TestcaseBegin("TEST6:Test for checking whether head is within bounds of
circular buffer");
	cbuff_status s = verify_ptr(test->head,test);
	UCUNIT_CheckIsEqual(s,ptr_valid); //PASS
	UCUNIT_TestcaseEnd();
}


/*****************************************************************************
***************
 * Function Name: test_cuff_resize(void)
 * Description :test for resizing of the buffer.
 * @input:void
 * @Return :void

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

//test-example-7
void test_cuff_resize(void)
{
	UCUNIT_TestcaseBegin("TEST7:Test for Resizing of Circular Buffer");
	cbuff_status s = cbuff_resize(test,20);
	printf(" No of Elements in Circular Buffer :: %d     ",test->size);
	UCUNIT_CheckIsEqual(s,cbuff_success); //PASS
	UCUNIT_TestcaseEnd();
}

/*****************************************************************************
***************
 * Function Name: test_cbuff_destroy(void)
 * Description :Test for destroying of Circular Buffer
 * @input:void
 * @Return :void

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

//test-example-8
void test_cbuff_destroy(void)
{
	UCUNIT_TestcaseBegin("TEST8:Test for Deletion of Circular Buffer");
	cbuff_status s =  cbuff_destroy(test);
	UCUNIT_CheckIsEqual(s,destroy_pass); //PASS
	UCUNIT_TestcaseEnd();
}



/*****************************************************************************
***************
 * Function Name:Testsuite_RunTests(void)
 * Description : this function runs all the ten tests in the test suite
 * @input:void
 * @Return :void
```

```
**********************************************************************************
**************/
void  Testsuite_RunTests(void)
{
       test_cbuff_init();
       test_cbuff_add();
       test_cbuff_wrap_add();
       test_cbuff_full();
       test_delete();
       test_head_ptr();
       test_cuff_resize();
       test_cbuff_destroy();
}

//int main(void)
//{
//    UCUNIT_Init();
//    UCUNIT_WriteString("\n************************************");
//    UCUNIT_WriteString("uCUnit demo application");
//    UCUNIT_WriteString(__DATE__);
//    UCUNIT_WriteString("\nTime:     ");
//    UCUNIT_WriteString(__TIME__);
//
//    UCUNIT_WriteString("\n************************************");
////    Testsuite_RunTests();
////    UCUNIT_Shutdown();
//
//    return 0;
//}
```