```
/*
File Name: circular_buffer.c
File Description: This file contains the implementation for
circular_buffer related functions.
Author Name: Nitik Satish Gupta and Rakesh Kumar
*/

#include "circular_buffer.h"
#include <stdint.h>
#include <stdlib.h>
#include <assert.h>
#include <stdio.h>
#include "uCUnit-v1.0.h"
#include "Logger.h"
#include "Status.h"
#include "led_control.h"
#define BUFFER_SIZE 5
extern uint16_t SIZE;

// Flag for init check
bool status_flag;

/*  Name: circular_buffer_init()
    Description: Function to carry out the initialization of the buffer
structure.
    Inputs: *buffer, size, cbuf_struct_handle
    Returns: void */
void circular_buffer_init(uint8_t * buffer, size_t size,
circularbuff_handle_t cbuf_struct_handle)
{     Log_String("\n\rINSIDE CIRCULAR BUFFER INITIALIZATION");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle);
      status_flag = circular_buffer_reset(cbuf_struct_handle);
      //could add test cases
      if(status_flag==FAIL)
      {
            led_control(Fail_case);
      }
      UCUNIT_CheckIsEqual(SUCCESS,status_flag);
      cbuf_struct_handle->buffer = buffer;
      cbuf_struct_handle->max = size;
      cbuf_struct_handle->full = 0;
      cbuf_struct_handle->tail = 0;
      cbuf_struct_handle->head = 0;
}

/*  Name: advance_pointer()
    Description: Function to handle the head increment and tail assignment
appropriately.
    Inputs: cbuf_struct_handle
    Returns: void */
static void advance_pointer(circularbuff_handle_t cbuf_struct_handle)
{     Log_String("\n\rINSIDE ADVANCE POINTER");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
```

```
      assert(cbuf_struct_handle);
      UCUNIT_CheckIsEqual(1,cbuf_struct_handle->full);
      if(1 == cbuf_struct_handle->full)
    {
            //cbuf_struct_handle->tail = (cbuf_struct_handle->tail + 1) %
cbuf_struct_handle->max;
            display_String("\n\rin advance_pointer full--> ");
            Print_Data(cbuf_struct_handle->full);
            cbuf_struct_handle->tail = (cbuf_struct_handle->tail + 1) %
SIZE;
    }

      //cbuf_struct_handle->head = (cbuf_struct_handle->head + 1) %
cbuf_struct_handle->max;
      cbuf_struct_handle->head = (cbuf_struct_handle->head + 1) % SIZE;
      display_String("\n\rcbuf_struct_handle->head = ");
      Print_Data(cbuf_struct_handle->head);

      cbuf_struct_handle->full = (cbuf_struct_handle->head ==
cbuf_struct_handle->tail);
      //display_String("\n\rout advance_pointer full--> ");
      Print_Data(cbuf_struct_handle->full);
}

/*  Name: circular_buffer_add()
    Description: Function to add an element into the circular buffer.
    Inputs: cbuf_struct_handle, data
    Returns: Error code */
Error circular_buffer_add(circularbuff_handle_t cbuf_struct_handle,
uint8_t data)
{     Log_String("\n\rINSIDE CIRCULAR BUFFER ADD");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      cbuf_struct_handle->buffer[cbuf_struct_handle->head] = data;
      display_String("\n\rChecking data inside the function
circular_buffer_add--> ");
      Print_Data(cbuf_struct_handle->buffer[cbuf_struct_handle->head]);
      advance_pointer(cbuf_struct_handle);
      UCUNIT_CheckIsEqual(1,cbuf_struct_handle->buffer[cbuf_struct_handle-
>head]);
      if(cbuf_struct_handle->buffer[cbuf_struct_handle->head])
      {
            return SUCCESS;
      }
      {
            led_control(Fail_case);
            return FAIL;
      }
}

/*  Name: retreat_pointer()
    Description: Function to move back the pointer for circular
implementation.
    Inputs: cbuf_struct_handle
    Returns: Error code */
```

```
static void retreat_pointer(circularbuff_handle_t cbuf_struct_handle)
{
      //Log_String("\n\rINSIDE RETREAT POINTER");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle);
      cbuf_struct_handle->full = 0;
      //cbuf_struct_handle->tail = (cbuf_struct_handle->tail + 1) %
cbuf_struct_handle->max;
      cbuf_struct_handle->tail = (cbuf_struct_handle->tail + 1) % SIZE;
}

/*  Name: circular_buffer_remove()
    Description: Function to remove an element from the circular buffer.
    Inputs: cbuf_struct_handle, *data
    Returns: uint8_t */
uint8_t circular_buffer_remove(circularbuff_handle_t cbuf_struct_handle,
uint8_t * data)
{
      Log_String("\n\rINSIDE CIRCULAR BUFFER REMOVE");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle && data && cbuf_struct_handle->buffer);
      int r = -1;
      if(!circular_buf_empty(cbuf_struct_handle))
      {
          *data = cbuf_struct_handle->buffer[cbuf_struct_handle->tail];
          retreat_pointer(cbuf_struct_handle);
          r = 0;
      }
      UCUNIT_CheckIsEqual(0,r);
      display_String("\n\rIn remove function: cbuf_struct_handle->tail =
");
      Print_Data(cbuf_struct_handle->tail);
      return r;
}

/*  Name: circular_buffer_reset()
    Description: Function to reset the circular buffer structure.
    Inputs: cbuf_struct_handle
    Returns: Error code */
Error circular_buffer_reset(circularbuff_handle_t cbuf_struct_handle)
{
      Log_String("\n\rINSIDE CIRCULAR BUFFER RESET");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
    assert(cbuf_struct_handle);
    cbuf_struct_handle->head = 0;
    cbuf_struct_handle->tail = 0;
    cbuf_struct_handle->full = 0;
    cbuf_struct_handle->count = 0;
    display_String("The buffer has been reset!!");
    return SUCCESS;
}

/*  Name: circular_buffer_full()
    Description: Function to check for buffer full condition.
```

```
    Inputs: cbuf_struct_handle
    Returns: Error code */
Error circular_buffer_full(circularbuff_handle_t cbuf_struct_handle)
{
      Log_String("\n\rINSIDE CIRCULAR BUFFER FULL");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle);
      UCUNIT_CheckIsEqual(1,cbuf_struct_handle->full);
      if(1 == (cbuf_struct_handle->full))
      {
            display_String("\n\rThe buffer is full, will loop-back and
overwrite henceforth!!");
            return SUCCESS;
      }
      else
      {
            //led_control(Fail_case);
            display_String("\n\rThe buffer still has some space!!");
      }
      return SUCCESS;
}

/*  Name: circular_buffer_capacity()
    Description: Function to check for buffer capacity.
    Inputs: cbuf_struct_handle
    Returns: size_t */
size_t circular_buffer_capacity(circularbuff_handle_t cbuf_struct_handle)
{
      Log_String("\n\rINSIDE circular_buffer_capacity");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle);
      return cbuf_struct_handle->max;
}

/*  Name: circular_buf_empty()
    Description: Function to check for buffer empty condition.
    Inputs: cbuf_struct_handle
    Returns: bool */
bool circular_buf_empty(circularbuff_handle_t cbuf_struct_handle)
{
      Log_String("\n\rINSIDE CIRCULAR BUFFER EMPTY");
      UCUNIT_CheckIsNotNull(cbuf_struct_handle);
      assert(cbuf_struct_handle);
    return (!(cbuf_struct_handle->full) && (cbuf_struct_handle->head ==
cbuf_struct_handle->tail));
}

/*  Name: circular_buffer_valid()
    Description: Function to check for valid buffer pointer memory
allocation.
    Inputs: cbuf_struct_handle
    Returns: bool */
bool circular_buffer_valid(circularbuff_handle_t cbuf_struct_handle)
{
```

```
	if (NULL == cbuf_struct_handle)
	{
		led_control(Fail_case);
		return 0;
	}
	else
	{
		return 1;
	}
}

/*  Name: circular_buffer_init_check()
    Description: Function to check for valid circular buffer
implementation.
    Inputs: cbuf_struct_handle
    Returns: bool */
bool circular_buffer_init_check(circularbuff_handle_t cbuf_struct_handle)
{
	if(!status_flag)
	{
		led_control(Fail_case);
	}
	return status_flag;
}

/*  Name: circular_buffer_destroy()
    Description: Function to destroy the circular buffer.
    Inputs: cbuf_struct_handle
    Returns: void */
void circular_buffer_destroy(circularbuff_handle_t cbuf_struct_handle)
{
	free(cbuf_struct_handle->buffer);
	free(cbuf_struct_handle);
	display_String("\n\rThe circular buffer has been successfully
destroyed!!");
}
/* File Name: led_control.c
File Description: This file contains implementation for LED initialize and
control
Author Name: Nitik Satish Gupta and Rakesh Kumar
*/

#include "Status.h"
#include <stdio.h>
#include <stdint.h>
#include "pin_mux.h"
#include "MKL25Z4.h"
#include <board.h>
#include "led_control.h"

/*    Name: delay()
	Description: This function provides a basic delay mechanism.
	Inputs: uint32_t
	Returns: None
```

```
*/
void delay()
{                uint16_t nof=1000;
            while(nof!=0) {
              __asm("NOP");
              nof--;
            }
}

/*    Name: Led_Initialize()
      Description: This function when called initializes the LED
appropriately.
      Inputs: None
      Returns: None
*/
void Led_Initialize()
{
           gpio_pin_config_t
led_pin_config1,led_pin_config2,led_pin_config3;
          led_pin_config1.pinDirection=kGPIO_DigitalOutput;
          led_pin_config1.outputLogic= 18u;
          GPIO_PinInit(GPIOB,18u,&led_pin_config1);
          led_pin_config2.pinDirection=kGPIO_DigitalOutput;
          led_pin_config2.outputLogic= 19u;
          GPIO_PinInit(GPIOB,19u,&led_pin_config2);
          led_pin_config3.pinDirection=kGPIO_DigitalOutput;
          led_pin_config3.outputLogic= 1u;
          GPIO_PinInit(GPIOD,1u,&led_pin_config3);
}

/*    Name: led_control()
      Description: This function is used to control the LED.
      Inputs: UART_State
      Returns: None
*/
void led_control(UART_State a)
{
      if(a==Initialization || a== Recieve)
      {
            LED_GREEN_OFF();
            LED_RED_OFF();
            LED_BLUE_ON();
            delay();
      }
      else if(a==Fail_case)
      {     LED_BLUE_OFF();
            LED_GREEN_OFF();
            LED_RED_ON();
            delay();
      }
      else if(a==Transmit )
      {     LED_RED_OFF();
            LED_GREEN_ON();
            LED_BLUE_OFF();
```

```
            delay();
        }
}

/*
 * 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 Name: Logger.c
File Description: This file contains implementation for the Logger
functionality
Author Name: Nitik Satish Gupta and Rakesh Kumar
*/
#include <stdio.h>
#include <stdint.h>
#include "board.h"
```

```
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "UARTFunction_Interrupt.h"
#include "UARTFunction_polled.h"
#include <math.h>
#include "Logger.h"
#include "Status.h"
#include "Time_Systick.h"
/* TODO: insert other include files here. */
#define DEMO_UART UART1                 //Using UART1 to implement UART
Operation
#define DEMO_UART_CLKSRC BUS_CLK  //Bus Clock for UART
#define DEMO_UART_CLK_FREQ CLOCK_GetFreq(BUS_CLK)   //For getting Bus
clock Frequency
#define multiplier 61
#define mod pow(2,15)
#define adder 7
#define Application 1
#define Echo 2
/* TODO: insert other definitions and declarations here. */
uint8_t Log_Status_Flag=0;

/*    Name: Log_Enable
      Description: This function enables the logging mechanism.
      Inputs: None
      Returns: None
*/
void Log_Enable()
{
    Log_Status_Flag=1;
}

/*    Name: Log_Disable
      Description: This function disables the logging mechanism.
      Inputs: None
      Returns: None
*/
void Log_Disable()
{
    Log_Status_Flag=0;
}

/*    Name: Log_Status
      Description: This function returns the logging status.
      Inputs: None
      Returns: None
*/
uint8_t Log_Status()
{
    return Log_Status_Flag;
}
```

```
/*    Name: Log_Data
      Description: This function logs numerical data.
      Inputs: uint32_t , size_t
      Returns: None
*/
void Log_Data(uint32_t *loc,size_t length)
{     uint8_t i;
      if(Log_Status_Flag)
      {
            for(i=0;i<length;i++)
            {
                  Transmit_polled(i);
            }
      }
}

/*    Name: Log_String
      Description: This function logs string data.
      Inputs: char str[]
      Returns: None
*/
void Log_String(char str[])
{
#ifndef NORMAL
      if(Log_Status_Flag)
      {
            display_String(str);
            display_time();
      }
#endif
}
/*    Name: displays_String
      Description: This function displays string data redirecting to UART
Function
      Inputs: string to be displayed
      Returns: None
*/
void display_String(char str[])
{
      if(Status==Polling)
      {
            Send_String_Poll(str);
      }
      else if(Status==Interrupt)
      {
            Send_String_interrupt(str);
      }
}
/*    Name: Log_Integer
      Description: This function logs integral data.
      Inputs: size_t
      Returns: None
*/
```

```
void Log_Integer(size_t a)
{
      if(Log_Status_Flag)
      {
            PRINTF("%d \n\r",a);

      }
}
/*    Name: Print_Data
      Description: This function displays data redirecting to UART
Function
      Inputs: data to be displayed
      Returns: None
*/
void Print_Data(uint8_t a)
{
      if(Status==Polling)
      {
            Transmit_polled(a);
      }
      else if(Status==Interrupt)
      {
            transmit_data_interrupt(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 Name: main.c
File Description: This is the main file that contains the primary function
calls.
Author Name: Nitik Satish Gupta and Rakesh Kumar
*/


#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 "UARTFunction_polled.h"
#include "UARTFunction_Interrupt.h"
#include "led_control.h"
#include "Logger.h"
#include "Status.h"
#include "Testsuite.h"
#include "System.h"
#include "circular_buffer.h"
#include "uCUnit-v1.0.h"
#include <stdlib.h>
#include <stdbool.h>
#include "Time_Systick.h"
#include "stdlib.h"
#define BUFFER_SIZE 5
uint16_t SIZE = BUFFER_SIZE;
uint8_t Status=Polling;
uint16_t count_chars[26] = {0};
uint16_t count_CHARS[26] = {0};
uint16_t count_num[10] = {0};
void display_report();
/* TODO: insert other include files here. */

/* TODO: insert other definitions and declarations here. */

/*
```

```
 * @brief   Application entry point.
 */
/*  Name: main()
    Description: This is the main function where all the respective
function calls are placed for the implementation
    Inputs: void
    Returns: int */
int main(void) {

  /* Init board hardware. */

  BOARD_InitBootPins();
    BOARD_InitBootClocks();
    BOARD_InitBootPeripherals();
  /* Init FSL debug console. */
    BOARD_InitDebugConsole();
    Led_Initialize();
    uart_init_polled();
    Log_Enable();

    //Configuring the systick below
    SysTick_Config(48000000/160000);
    //uart_init_interrupt();
    uint8_t data = 0;

    //Setting the required arrays to 0 for clearing garbage
    memset(count_CHARS,0,sizeof(count_CHARS));
    memset(count_chars,0,sizeof(count_chars));
    memset(count_num,0,sizeof(count_num));

    uint8_t choice;
    display_String("Hello User,There are two modes you can work in.\n\r");
    display_String("1.Polling\n\r");
    display_String("2.Interrupt");
    display_String("\n\r Please enter the Mode You want to work in:");
        choice=Recieve_polled();
        Transmit_polled(Status);
        choice=choice-48;
        if(choice==1)
        {
           display_String("\n\r Starting with Polling mode.");
           Status=Polling;
        }
        else if(choice==2)
        {
           display_String("\n\r Starting Interrupt mode.");
           uart_init_interrupt();
           Status=Interrupt;
        }
        else
        {
           display_String("\n\r Sorry Wrong choice inserted.");
           display_String("\n\r Starting with Polling mode.");
        }
```

```
        display_String("\n\r=== Circular Buffer Initialization ===\n");
        //printf("\n\rThe size of the initialized structure is --
>%d\n\r",cbuf_handle->max);
        //printf("\n\r******\n\rAdding continuous data%d values\n\r",
BUFFER_SIZE);
      //circular_buffer_add(cbuf_handle, 5);

        int32_t i=-1;
    /////For input from UART
        uint8_t char_array[BUFFER_SIZE] = {'\0'};
        //uint32_t iter = 0;
        do
        {   i++;
        display_String("\n\r Give Values");
            if(Status==Polling)
            {
                  char_array[i] = Recieve_polled();
                  Print_Data(char_array[i]);
            }
            else if(Status==Interrupt)
            {
                  char_array[i] = Receive_data_interrupt();
                  Print_Data(char_array[i]);
            }
        }while((int)char_array[i]!=13);
        /////For input from UART
    //        for(i=0;i<5;i++)
    //        {
    //           printf("\n\r%c",char_array[i]);
    //        }
             uint8_t * buffer = malloc(SIZE * sizeof(uint8_t));
             assert(buffer);
             circularbuff_handle_t cbuf_handle = malloc(SIZE *
sizeof(circular_buffer_t));

             bool res = circular_buffer_valid(cbuf_handle);
             if(res)
             {
                  display_String("\n\r The buffer pointer is valid");
             }
             else
             {
                  display_String("\n\r Invalid Buffer pointer");
             }

             circular_buffer_init(buffer, SIZE, cbuf_handle);

             res = circular_buffer_init_check(cbuf_handle);
             if(res)
             {
                  display_String("\n\r The buffer pointer is initialized");
             }
             else
```

```
            {
                 display_String("\n\r Couldn't initialize buffer");
            }

            cbuf_handle->tail = 0;

            //Adding the data to the circular buffer
            for(uint32_t i = 0; i < 7; i++)
            {
                circular_buffer_add(cbuf_handle, char_array[i]);
                display_String("\n\rData being added char_array[i]: ");
                Print_Data(char_array[i]);
                circular_buffer_full(cbuf_handle);
            }
    //        for(uint8_t i = 0; i < (SIZE+2); i++)
    //        {
    //           PRINTF("\n\rEnter data to be entered into the circular
buffer\n\r");
    //
    //           scanf("%c",&data);
    //           circular_buffer_add(cbuf_handle, data);
    //           circular_buffer_full(cbuf_handle);
    //        }

           // uint8_t data;
            //circular_buffer_remove(cbuf_handle, &data);

            //Removing oldest element from the circular buffer
            for(uint8_t k = 0; k < (SIZE); k++)
            {
                 circular_buffer_remove(cbuf_handle, &data);
                 //Send_String_Poll("\n\rThe last data in the buffer
that's read and removed is ");
                display_String("\n\rdata-> ");
                Print_Data(data);
                 //Transmit_polled(data);
            }
           //circular_buffer_remove(cbuf_handle, &data);

            //display_String("\n\rEnter a string to check for character
frequency: ");

            uint32_t a = 0;

            //Logic to maintain count for the entered characters
            for(a = 0; a < sizeof(char_array); a++)
            {
                 if((char_array[a] >= 'a') && (char_array[a] <= 'z'))
                 {
                       count_chars[char_array[a] - 'a']++;
                 }
                 if((char_array[a] >= 'A') && (char_array[a] <= 'Z'))
                 {
                       count_CHARS[char_array[a] - 'A']++;
```

```
                }
                if((char_array[a] >= '0') && (char_array[a] <= '9'))
                {
                      count_num[char_array[a] - '0']++;
                }
           }

           // Displaying the report that was populated
           display_report();
           //}

           //Destroying the circular buffer
           circular_buffer_destroy(cbuf_handle);
        while(1) {
           //Just an infinite wait;
        }
        return 0 ;
    /* Force the counter to be placed into memory. */
    return 0 ;
}


/*  Name: display_report()
    Description: This function displays the report.
    Inputs: void
    Returns: void */
void display_report()
{    char cha[10];
    uint32_t a = 0;

    //Frequency for lower-case letters
      for(a='a';a<='z';a++)
      {
            if(count_chars[a-'a']!=0)
            {
                  display_String("\n\rFrequency of ");
                  Print_Data(a);
                  display_String("is");
                  itoa(count_chars[a-'a'],cha,10);
                  display_String(cha);
            }
      }

      //Frequency for upper-case letters
      for(a='A';a<='Z';a++)
      {
            if(count_CHARS[a-'A']!=0)
            {
                  display_String("\n\rFrequency of ");
                  Print_Data(a);
                  display_String("is");
                  itoa(count_CHARS[a-'A'],cha,10);
                  display_String(cha);
            }
```

```
	}

	//Frequency for numbers
	for(a='0';a<='9';a++)
	{
		if(count_num[a-'0']!=0)
		{

			display_String("\n\rFrequency of ");
			Print_Data(a);
			display_String(" is ");
			itoa(count_num[a-'0'],cha,10);
			display_String(cha);
		}
	}
}


/*
 * File Name:Time_Systick.c
 * Description: The Time_systick.c contains the interrupt function and the
display function for time elapsed during the execution
 *  Created on: Nov 18, 2019
 *      Author: Nitik gupta and Rakesh Kumar
 */
#include "Time_Systick.h"
#include "Logger.h"
#include "stdlib.h"
#include <stdint.h>
volatile unsigned int t=0,S=0,M=0,H=0,mil=0;
char Hours[10],Mins[10],Seconds[10],Milli[10];
/*    Name: SysTick_Handler()
      Description: The IRQ Handler for the Systic Function
      Inputs: None
      Returns: None
*/
void SysTick_Handler()
{
        t++;
        if(t%160 == 0) // After every 160 clock, one millisecond will be
updated
        {
           mil++;
           t=0;
        }
        if(mil==1000)
        {
           S++;       //After every 1000 milliseconds 1 second will be
incremented
           mil=0;
        }
}
/*    Name: display_time()
      Description: The display function which displays time of the systic
```

```
      Inputs: None
      Returns: None
*/
void display_time()
{     M = S/60;
      S = S%60;
      H = M/60;
      M = M%60;
      uitoa(mil, Milli, 10);
      uitoa(S, Seconds, 10);
      uitoa(M, Mins , 10);
      uitoa(H, Hours, 10);
      display_String("\t");
      display_String(Hours);
      display_String(":");
      display_String(Mins);
      display_String(":");
      display_String(Seconds);
      display_String(".");
      display_String(Milli);

}
/*
 * File Name: UARTFunction_Interrupt.c
 *    Description: This .c contains all the functions required to make the
Program work in interrupt mode.
 *  Created on: Nov 15, 2019
 *  Author: Nitik Gupta and Rakesh Kumar
 */
#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 <stdint.h>
#include "Logger.h"
#include "Status.h"
#include "UARTFunction_Interrupt.h"
#include "UARTFunction_polled.h"
#include "led_control.h"
#define BaudRate 115200           //Keeping Baudrate at 115200
volatile uint8_t character_recieve,character_transmit;
volatile uint8_t flag=0;
volatile uint8_t txavailable=0;
/*    Name: delay2()
      Description: This function provides a basic delay mechanism.
      Inputs: uint32_t
      Returns: None
*/
void delay2()
{
      for(uint16_t j=0;j<1000;j++)
```

```
	{
		__asm("NOP");
	}
}
/*	Name: uart_init_interrupt()
	Description: This function when called initializes the Interrupt
based UART appropriately.
	Inputs: None
	Returns: None
*/
void uart_init_interrupt()
{	led_control(Initialization);
	Log_String("\n\rINSIDE INITIALIZATION OF INTERRUPT");
	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/2)/(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);

	// Enable interrupts. Listing 8.11 on p. 234
	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);
	// 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;

}
/*	Name: transmit_available_interrupt()
	Description: Checks whether transmit is available or not
	Inputs: None
	Returns: Integer that show it is available or not
*/
uint8_t transmit_available_interrupt()
{	led_control(Transmit);

	UART0->C2|= UART0_C2_TIE_MASK;

	if(txavailable)
	//(UART0->S1 & UART0_S1_TDRE_MASK)
	{
		return 1;
	}
	else
	{
		return 0;
	}
}
/*	Name: transmit_data_interrupt()
	Description: Transmits data to Uart
	Inputs: Value to be transmitted
	Returns: None
*/
void transmit_data_interrupt(uint8_t ch)
{	led_control(Transmit);
	if(transmit_available_interrupt())
	{	uint8_t temp=UART0->D;
		UART0->D=ch;
		txavailable=0;
	}
```

```
	delay2();
}
/*	Name: Receive_available_interrupt()
	Description: Checks whether receive is available or not
	Inputs: None
	Returns: Integer that show it is available or not
*/
uint8_t Receive_available_interrupt()
{	led_control(Recieve);
	if(UART0->S1 && UART0_S1_RDRF_MASK)
	{
		return 1;
	}
	else
	{
		return 0;
	}
}
/*	Name: Receive_data_interrupt()
	Description: receives data from Uart
	Inputs: Value to be receives
	Returns: None
*/
uint8_t Receive_data_interrupt()
{	led_control(Recieve);
	uint8_t temp=UART0->D;
	UART0->C2 |= UART0_S1_RDRF_MASK;
	flag=2;
	PRINTF("\n\r ENTER a Character:");
	while(1)
	{

		if(flag==1)
		{	UART0->S1 &= ~UART0_S1_RDRF_MASK;
			break;

		}
	}
	return character_recieve;
}
/*	Name: Send_String_interrupt()
	Description: Sends the string value to the interrupt
	Inputs: Value to be sent
	Returns: None
*/
void Send_String_interrupt(char * str) {
	// enqueue string
	while (*str != '\0') { // Send characters up to null terminator
		character_transmit=*str++;
		transmit_data_interrupt(character_transmit);
	}
	UART0->C2&=~UART0_C2_TIE_MASK;
}
/*	Name: UART0_IRQHandler(void)
```

```
      Description: Interrupt handler for the Interrupt mode
      Inputs: None
      Returns: None
*/
void UART0_IRQHandler(void) {
      printf("\n\rINSIDE ISR");
      delay2();
      if (UART0->S1 & (UART_S1_OR_MASK |UART_S1_NF_MASK |
            UART_S1_FE_MASK | UART_S1_PF_MASK))
      {
                  // clear the error flags
                  UART0->S1 |= UART0_S1_OR_MASK | UART0_S1_NF_MASK |
                                                      UART0_S1_FE_MASK |
UART0_S1_PF_MASK;
                  // read the data register to clear RDRF
                  //character_recieve = UART0->D;
      }
      if (Receive_available_interrupt())
      {
                                    // received a character
                                    character_recieve = UART0->D;
                                    if(flag==2)
                                    {

      character_transmit=character_recieve;
                                          flag=1;
                                    }
                                    //PRINTF("%d",character_recieve);
                                    //delay2();
      }
      if ( UART0->S1 & UART0_S1_TDRE_MASK)
            { // tx buffer empty
            // can send another character
                  txavailable=1;
                  UART0->C2&= ~UART0_C2_TIE_MASK;


            }



}
//FILE __stdout;  //Use with printf
//FILE __stdin;        //use with fget/sscanf, or scanf
//

//Retarget the fputc method to use the UART0
//int fputc(int ch, FILE *f){
//    while(!(UART0->S1 & UART_S1_TDRE_MASK) && !(UART0->S1 &
UART_S1_TC_MASK));
//    UART0->D = ch;
//    return ch;
//}
//
```

```
////Retarget the fgetc method to use the UART0
//int fgetc(FILE *f){
//    while(!(UART0->S1 & UART_S1_RDRF_MASK));
//    return UART0->D;
//}
//
/*
 * File Name: UARTFunction_polled.c
 *    Description: This .c contains all the functions required to make the
Program work in interrupt mode.
 *  Created on: Nov 15, 2019
 *  Author: Nitik Gupta and Rakesh Kumar
 */
#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 "UARTFunction_Interrupt.h"
#include "Logger.h"
#include "Status.h"
#include <stdint.h>
#include "led_control.h"
#include "UARTFunction_polled.h"
FILE __stdout;  //Use with printf
FILE __stdin;    //use with fget/sscanf, or scanf
/*    Name: delay1()
      Description: This function provides a basic delay mechanism.
      Inputs: uint32_t
      Returns: None
*/
void delay1()
{
      for(uint16_t j=0;j<1000;j++)
      {
            __asm("NOP");
      }
}
/*    Name: uart_init_polled()
      Description: This function when called initializes the Polling based
UART appropriately.
      Inputs: None
      Returns: None
*/
void uart_init_polled()
{     led_control(Initialization);
      Log_String("\n\rINSIDE INITIALIZATION OF POLLED");

      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/2)/(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);
NVIC_DisableIRQ(UART0_IRQn);
UART0->C2 |= UART_C2_RIE(0)|UART_C2_TIE(0);
      UART0->C2|=UART0_C2_TE_MASK  | UART0_C2_RE_MASK ;
      temp = UART0->D;
      UART0->S1 &= ~UART0_S1_RDRF_MASK;

}
/*    Name: transmit_available()
      Description: Checks whether transmit is available or not
      Inputs: None
      Returns: Integer that show it is available or not
```

```
*/
uint8_t transmit_available()
{     led_control(Transmit);
      while(!(UART0->S1 & UART0_S1_TDRE_MASK));
            return 1;

}
/*    Name: transmit_data()
      Description: Transmits data to Uart
      Inputs: Value to be transmitted
      Returns: None
*/
void transmit_data(uint16_t ch)
{     led_control(Transmit);
            UART0->D=ch;
}
/*    Name: transmit_polled()
      Description: Polling based Transmit
      Inputs: Value to be transmitted
      Returns: None
*/
void Transmit_polled(uint16_t ch)
{     led_control(Transmit);
      if(transmit_available())
      {
            transmit_data(ch);
      }
}
/*    Name: Receive_available()
      Description: Checks whether receive is available or not
      Inputs: None
      Returns: Integer that show it is available or not
*/
uint8_t Receive_available()
{     led_control(Recieve);
      while(!(UART0->S1 & UART0_S1_RDRF_MASK));
      return 1;
}
/*    Name: Receive_data()
      Description: receives data from Uart
      Inputs: Value to be receives
      Returns: None
*/
uint8_t Receive_data()
{     led_control(Recieve);
            return UART0->D;
}
/*    Name: transmit_polled()
      Description: Polling based Recieve function
      Inputs: Value to be transmitted
      Returns: None
*/
char Recieve_polled()
{     led_control(Recieve);
```

```
	uint8_t temp=UART0->D;
	char ch;
	if(Receive_available())
	{
		ch=UART0->D;
	}
	return ch;
}
//Retarget the fputc method to use the UART0
int fputc(int ch, FILE *f)
{
	while(!(UART0->S1 & UART_S1_TDRE_MASK) && !(UART0->S1 &
UART_S1_TC_MASK));
	UART0->D = ch;
	return ch;
}

//Retarget the fgetc method to use the UART0
int fgetc(FILE *f)
{
	while(!(UART0->S1 & UART_S1_RDRF_MASK));
	return UART0->D;
}
/*	Name: Send_String_polled()
	Description: Sends the string value to the polling mode
	Inputs: Value to be sent
	Returns: None
*/
void Send_String_Poll(uint8_t * str) {
	while (*str != '\0') { // Send characters up to null terminator
		Transmit_polled(*str++);
		delay1();
	}
}
```