```c
/*
 * Created By : Vaidehi Salway & Kushagra Pandey
 * Date : 04/11/2019
 * TMP102 interfaced to KL25Z freedom board Using I2C
 * main.c
 *
 *
 */
#include "Unit_Testing.h"
#include "UART.h"

//echo mode code
#ifdef Echo_Mode
#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "Logger.h"
#include "LED_Blink.h"

int8_t RX_Interrupt_Flag=0;
int8_t TX_Interrupt_Flag=0;

//unit testing fucntion calls
#ifdef Unit_Test
	Test_Data_Access();
	Wrap_Remove();
	Wrap_Add();
	Buffer_OverFill();
	Buffer_OverEmpty();
	Buffer_Destroy();
#endif


int main()
{

	/* Init board hardware. */
	BOARD_InitBootPins();
	BOARD_InitBootClocks();
	BOARD_InitBootPeripherals();
	/* Init FSL debug console. */
	BOARD_InitDebugConsole();
	BOARD_InitPins();
	BOARD_BootClockRUN();
//init UART with a baudrate
	Init_UART0(115200);

	//initialize all the LEDS
	RED_LED_INIT();
	BLUE_LED_INIT();
	GREEN_LED_INIT();
```

```
	BLUE_LED_ON();
	Delay_Time(1000);
	BLUE_LED_OFF();
	Delay_Time(200);

	//code block for the interrupt mode
#if USE_UART_INTERRUPTS==0 // Polling version of code

	rx_cb=malloc(LENGTH*sizeof(Buffer_Parameters));
	Init_Buffer(rx_cb,LENGTH);

	while (1)
	{

		//Poling_Function();
		int8_t ch;
		while (!(UART0->S1 & UART0_S1_RDRF_MASK))
			;
		ch = UART0->D;
		Echo(ch);
	}


//code block for the Polling mode
#elif USE_UART_INTERRUPTS==1 // Interrupt version of code

	tx_cb=malloc(LENGTH*sizeof(Buffer_Parameters));
	Init_Buffer(tx_cb,LENGTH);
	int8_t ch;

	while (1)
	{

		UART0->C2 &= ~(UART0_C2_TIE(1));

		if(RX_Interrupt_Flag==1)
		{
			ch = UART0->D;
			RX_Interrupt_Flag=0;
			Echo(ch);

		}

		//enable TX
		if (!(UART0->C2 & UART0_C2_TIE_MASK))
		{
			UART0->C2 |= UART0_C2_TIE(1);
		}

	}

#endif
```

```
	return 0;

}
#endif

//Application mode code
#ifdef Application_Mode

#include "board.h"
#include "peripherals.h"
#include "pin_mux.h"
#include "clock_config.h"
#include "MKL25Z4.h"
#include "fsl_debug_console.h"
#include "Logger.h"
#include "Delay_Function.h"
#include "LED_Blink.h"

int8_t RX_Interrupt_Flag=0;
int8_t TX_Interrupt_Flag=0;



int main()
{

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

	Init_UART0(115200);
	Init_Systick();

	//initialize all the LEDS
	RED_LED_INIT();
	BLUE_LED_INIT();
	GREEN_LED_INIT();


	BLUE_LED_ON();
	Delay_Time(200);
	BLUE_LED_OFF();
	Delay_Time(100);


//unitt test function calls
#ifdef Unit_Test
	Test_Data_Access();
	Wrap_Remove();
	Wrap_Add();
```

```
	Buffer_OverFill();
	Buffer_OverEmpty();
	Buffer_Destroy();
#endif

#if USE_UART_INTERRUPTS==0 // Polling version of code

	rx_cb=malloc(LENGTH*sizeof(Buffer_Parameters));
	Init_Buffer(rx_cb,LENGTH);

	Log_String(0,1,"POLLING_MODE");

	while (1)
	{
		Buffer_Display();

	}

#elif USE_UART_INTERRUPTS==1 // Interrupt version of code


	rx_cb=malloc(LENGTH*sizeof(Buffer_Parameters));
	Init_Buffer(rx_cb,LENGTH);

	Log_String(0,1,"INTERRUPT_MODE");
	while (1)
	{
		START_CRITICAL();

		//disabling transmitter
		UART0->C2 &= ~(UART0_C2_TIE(1));

		if(RX_Interrupt_Flag==1)
		{
			Buffer_Display();
			RX_Interrupt_Flag=0;
		}

		//enable TX
		if (!(UART0->C2 & UART0_C2_TIE_MASK))
		{
			UART0->C2 |= UART0_C2_TIE(1);
		}
		END_CRITICAL();
	}

#endif

	return 0;
}
#endif
```

```c
/*
 * @file UART.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include "UART.h"
#include "Counting_Characters.h"
#include "LED_Blink.h"

	Buffer_Parameters* rx_cb=NULL;
	Buffer_Parameters* tx_cb=NULL;

	extern int8_t RX_Interrupt_Flag;
	extern int8_t TX_Interrupt_Flag;
	int8_t ch;

	int8_t full_flag=0;
	int8_t empty_flag=0;


	// Code listing 8.8, p. 231
	void Init_UART0(uint32_t baud_rate)
	{

		uint16_t sbr;
		uint16_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);

      #if USE_UART_INTERRUPTS
            // 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);
      #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;

      }

      int8_t ch;

      // UART0 IRQ Handler. Listing 8.12 on p. 235
      void UART0_IRQHandler(void)
      {
            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
                  ch = UART0->D;

            }
```

```
		if (UART0->S1 & UART0_S1_RDRF_MASK)
		{
			// received a character
			RX_Interrupt_Flag=1;

#ifdef Echo_Mode
			ch = UART0->D;
			RX_Interrupt_Flag=0;
			Echo(ch);

#endif
#ifdef Application_Mode
			Buffer_Display();
#endif
		}

		if ((UART0->C2 & UART0_C2_TIE_MASK) && // transmitter interrupt enabled
			(UART0->S1 & UART0_S1_TDRE_MASK))
		{
			TX_Interrupt_Flag=1;
		}
		else
		{
			// queue is empty so disable transmitter interrupt
			UART0->C2 &= ~UART0_C2_TIE_MASK;
		}

	}

	//function to transmit a chracter to the terminal
	void UART0_Transmit_Poll(int8_t data)
	{
		while (!(UART0->S1 & UART0_S1_TDRE_MASK))
			;
		UART0->D = data;
	}

	//function to receive the vale on the terminal
	int8_t UART0_Receive_Poll(void)
	{

		BLUE_LED_ON();
		Delay_Time(20);
		BLUE_LED_OFF();

		int8_t ch;
		while (!(UART0->S1 & UART0_S1_RDRF_MASK))
			;
		ch = UART0->D;
		Add_Element_To_Buffer(rx_cb,ch);

		return ch;
	}
```

```
#ifdef Application_Mode
	//adding elements received on terminal to the circular buffer in application
mode
	int8_t Receive_Data_Interupt()
	{
		ch = UART0->D;
		Add_Element_To_Buffer(rx_cb,ch);

		return ch;
	}
#endif

//get the value in the receive buffer of UART in Echo_mode
#ifdef Echo_Mode

	int8_t Receive_Data_Interupt()
	{

		GREEN_LED_ON();

		ch = UART0->D;

		return ch;
	}

#endif


#ifdef Echo_Mode
	//retransmit the value received to the terminal in Echo mode
	void Echo(int8_t d)
	{
		char sp=' ';

		GREEN_LED_ON();


//check if the buffer is full, if yes then stop echoing
		if(!(full_flag))
		{
			UART0_Transmit_Poll(d);
		}
		else
		{
			UART0_Transmit_Poll(sp);
		}
	}
#endif


//get the value in the receive buffer of UART in Apllication_Mode
#ifdef Application_Mode
	void Echo(int8_t d)
	{
		char sp=' ';
```

```
			GREEN_LED_ON();
			Delay_Time(20);
			GREEN_LED_OFF();

//check if the buffer is full, if yes then stop echoing
			if(!(full_flag))
			{
				UART0_Transmit_Poll(d);

			}
			else
			{
				UART0_Transmit_Poll(sp);
			}

	}
#endif


#ifdef Echo_Mode
	void Poling_Function()
	{
			int8_t c;

			Check_Buffer(rx_cb,LENGTH);
			c = UART0_Receive_Poll();
			Echo(c);

			//wrap around condition while removing
			if(full_flag)
			{
				Remove_Element_From_Buffer(rx_cb,6);
			}

	}
#endif

#ifdef Application_Mode
//receiving and retransmitting the value to the terminal
	int8_t Poling_Function()
	{
			Time_Stamp();
			BLUE_LED_ON();
			Delay_Time(20);
			BLUE_LED_OFF();
			int8_t c;
			c = UART0_Receive_Poll();
			Echo(c);

			return c;
	}

//receiving and retransmitting the value to the terminal in the buffer_expand mode
#if Buffer_Expand==1
```

```
	void Buffer_Display()
	{
		Time_Stamp();

		int8_t Val[LENGTH]={0};
		Log_String(0,1,"Buffer:");
		int z=1;


		for(int i=0; i<LENGTH; i++)
		{
			Val[i]=Poling_Function();
			PRINTF("\n\rASCII Value:%d\n\r",Val[i]);
			Count_Characters(Val,z);
			z++;
		}

	}


#elif Buffer_Expand==0
//get the chracter count in application mode
	void Buffer_Display()
	{
		Time_Stamp();
		Poling_Function();
		if(full_flag)
		{
			Expand_Buffer(rx_cb,6);
			full_flag=0;
		}
	}
#endif

#endif
```

```
/*
 * @file UART.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include <MKL25Z4.H>

#include "CIRCULAR_BUFFER.h"
#include "Delay_Function.h"

//#define Application_Mode
#define Echo_Mode
//#define Unit_Test
```

```
#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)

//size of the buffer
#define LENGTH  30


#define START_CRITICAL()  __disable_irq()
#define END_CRITICAL()    __enable_irq()

//Initialize UART
void Init_UART0(uint32_t baud_rate);
void UART0_Transmit_Poll(int8_t data);
int8_t UART0_Receive_Poll(void);

int8_t Receive_Data_Interupt();
void Echo(int8_t d);
void Echo1(int8_t *d,size_t length);

#ifdef Echo_Mode
void Poling_Function();
#endif

//function definations for application mode
#ifdef Application_Mode
int8_t Poling_Function();
void Buffer_Display();
#endif

void Poling_Function1();

void Display_Data(int8_t* data);


//pointers for dynamic allocation of memory
extern Buffer_Parameters* rx_cb;
extern Buffer_Parameters* tx_cb;
extern  Buffer_Parameters* ut_cb;



/*
 * @file CIRCULAR_BUFFER.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include "CIRCULAR_BUFFER.h"
#include "fsl_uart.h"
#include "fsl_debug_console.h"
#include "LED_Blink.h"
```

```
extern int8_t full_flag;
extern int8_t empty_flag;

extern int8_t Null_ptr;
extern int8_t flag_full_ut;
extern int8_t flag_empty_ut;



//Initializing the pointer for Circular buffer by allocating memory.
BUFFER_STATUS Init_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length)

{


	if(Buffer_Ptr==NULL || Buffer_Length<=0)
	{
		Null_ptr=1;
		return BUFFER_POINTER_NULL;
	}

	else
	{

		Buffer_Ptr-
>Buffer_Base_Pointer=(int8_t*)malloc(Buffer_Length*(sizeof(char)));

		Buffer_Ptr->Buffer_Head=Buffer_Ptr->Buffer_Base_Pointer;
		Buffer_Ptr->Buffer_Tail=Buffer_Ptr->Buffer_Head;
		Buffer_Ptr->Buffer_Length=Buffer_Length;
		Buffer_Ptr->Range=Buffer_Ptr->Buffer_Base_Pointer+(Buffer_Length-1);
		Buffer_Ptr->Buffer_Count=0;

		return BUFFER_INIT_SUCCESS;

	}
}
//Checking if the Circular buffer is EMPTY or is FULL
BUFFER_STATUS Check_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length)

{
//check if the buffer counter is null
	if(Buffer_Ptr==NULL)
	{
		Null_ptr=1;
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();



		return BUFFER_POINTER_NULL;
	}
```

```
	//check if the buffer is empty
	if(Buffer_Ptr->Buffer_Count==0)
	{
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();


		empty_flag=1;
		flag_empty_ut=1;
		return BUFFER_IS_EMPTY;
	}
	//check if the buffer is full
	else if(Buffer_Ptr->Buffer_Count>=Buffer_Ptr->Buffer_Length)
	{
		Log_String(0,1," BUFFER IS FULL");
		full_flag=1;
		flag_full_ut=1;
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();

		return BUFFER_IS_FULL;

	}
	return BUFFER_INIT_SUCCESS;

}

//Adding a single element to the Circular buffer
BUFFER_STATUS Add_Element_To_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Value)

{

	//check if the buffer counter is null
	if(Buffer_Ptr==NULL)
	{
		Null_ptr=1;
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();

		return BUFFER_POINTER_NULL;
	}

	else
	{
		//check if the buffer is full
		if(Buffer_Ptr->Buffer_Count>=Buffer_Ptr->Buffer_Length)
		{
			Log_String(0,1,"BUFFER IS FULL");
			full_flag=1;
			flag_full_ut=1;
			RED_LED_ON();
```

```
                Delay_Time(20);
                RED_LED_OFF();


                return BUFFER_IS_FULL;
        }

        else
        {
                BLUE_LED_ON();
                Delay_Time(20);
                BLUE_LED_OFF();


                if(Buffer_Ptr->Buffer_Tail==Buffer_Ptr->Range)
                {

                        *(Buffer_Ptr->Buffer_Head)=Value;
                        Buffer_Ptr->Buffer_Head+=1;

                }
                else
                {
                        *(Buffer_Ptr->Buffer_Head)=Value;
                        Buffer_Ptr->Buffer_Head+=1;
                }

        }
        Buffer_Ptr->Buffer_Count++;
        PRINTF("\n\rcount %d\n\r",Buffer_Ptr->Buffer_Count);

        return BUFFER_INIT_SUCCESS;
    }

}


//Adding a single element to the Circular buffer
BUFFER_STATUS Add_String_To_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t * Data,
int8_t Length)
{
    //check if the buffer counter is null
    if(Buffer_Ptr==NULL)
    {
        Null_ptr=1;
        RED_LED_ON();
        Delay_Time(20);
        RED_LED_OFF();


        return BUFFER_POINTER_NULL;
    }

    else
    {
```

```
		int i;
		BUFFER_STATUS Last_Data;
		for(i=0;i<Length;i++)
		{
			Last_Data=Add_Element_To_Buffer(Buffer_Ptr,*(Data+i));

		}
		return Last_Data;

	}

}



//Removing elements from the Circular Buffer
BUFFER_STATUS Remove_Element_From_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Value)
{
	//check if the buffer counter is null
	if(Buffer_Ptr==NULL)
	{
		Null_ptr=1;
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();

		return BUFFER_POINTER_NULL;
	}

	else
	{
		//check if the buffer is empty
		if(Buffer_Ptr->Buffer_Count==0)
		{
			empty_flag=1;
			flag_empty_ut=1;
			RED_LED_ON();
			Delay_Time(20);
			RED_LED_OFF();

			Log_String(0,1," BUFFER IS EMPTY");
			return BUFFER_IS_EMPTY;
		}

		Value=*(Buffer_Ptr->Buffer_Tail);

		if(Buffer_Ptr->Buffer_Tail==Buffer_Ptr->Range)
		{
			Buffer_Ptr->Buffer_Tail=Buffer_Ptr->Buffer_Base_Pointer;

		}
		else
		{
			Buffer_Ptr->Buffer_Tail++;
		}
```

```
		Buffer_Ptr->Buffer_Count--;
		PRINTF("\n\rcount:%d\n\r",Buffer_Ptr->Buffer_Count);

		return BUFFER_INIT_SUCCESS;
	}


}

BUFFER_STATUS Destroy_Buffer(Buffer_Parameters *Buffer_Ptr)
{

	//check if the buffer counter is null
	if(Buffer_Ptr==NULL)
	{
		Null_ptr=1;
		RED_LED_ON();
		Delay_Time(20);
		RED_LED_OFF();

		return BUFFER_POINTER_NULL;
	}

	else
	{
		free(Buffer_Ptr->Buffer_Base_Pointer);
		return BUFFER_INIT_SUCCESS;
	}


}

#ifdef Buffer_Expand
//fucntion to expand the buffer size once the bufefr is full, using realloc().
BUFFER_STATUS Expand_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length)
{

	Log_String(0,1," expanding buffer size");

	Buffer_Ptr->Buffer_Base_Pointer=(int8_t*)realloc(Buffer_Ptr-
>Buffer_Base_Pointer,Buffer_Length);

			Buffer_Ptr->Buffer_Head=Buffer_Ptr->Buffer_Base_Pointer;
			Buffer_Ptr->Buffer_Tail=Buffer_Ptr->Buffer_Head;
			Buffer_Ptr->Buffer_Length=Buffer_Length;
			Buffer_Ptr->Range=Buffer_Ptr->Buffer_Base_Pointer+(Buffer_Length-
1);
			Buffer_Ptr->Buffer_Count=0;

			return BUFFER_INIT_SUCCESS;

}
#endif
```

```c
/*
 * @file CIRCULAR_BUFFER.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "Logger.h"

#define START_CRITICAL1()  __enable_irq()
#define END_CRITICAL1()    __disable_irq()

#define Buffer_Expand   (1)//1 for non-expandable buffer, 0 for expandable buffer

//returning status of the circular buffer using enums
typedef enum
{
	BUFFER_IS_EMPTY,
	BUFFER_IS_FULL,
	POINTER_INIT_ERROR,
	BUFFER_POINTER_NULL,
	BUFFER_INIT_SUCCESS

}BUFFER_STATUS;

//parameters for the circular bufer are defined in a structure
 typedef struct//pointer, head, tail, length, count
{
	int8_t* Buffer_Base_Pointer;
	int8_t* Buffer_Head ;
	int8_t* Buffer_Tail;
	size_t   Buffer_Length;
	int8_t  Buffer_Count;
	int8_t* Range;

}Buffer_Parameters;


BUFFER_STATUS Init_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length);

BUFFER_STATUS Check_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length);

BUFFER_STATUS Add_Element_To_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Value);

BUFFER_STATUS Add_String_To_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t * Data,
int8_t Length);

BUFFER_STATUS Remove_Element_From_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t
Value);
```

```
BUFFER_STATUS Destroy_Buffer(Buffer_Parameters *Buffer_Ptr);

#ifdef Buffer_Expand
BUFFER_STATUS Expand_Buffer(Buffer_Parameters *Buffer_Ptr, int8_t Buffer_Length);
#endif
```

```
/*
 * @file Counting_Characters.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include "Counting_Characters.h"
#include "UART.h"

//function to get the count for different characters
void Count_Characters(int8_t Buffer_char[LENGTH3], int count_count)
{

  int c = {0}, count[128] = {0}, x, counted = 0,i=0;
for(int i=0;i<LENGTH3;i++)
{
    if(Buffer_char[i] >= '!' && Buffer_char[i] <= '~')
    {
      x = Buffer_char[i] - 33;
      count[x]++;
    }

    c++;

}
//display only after the buffer is full
 if(count_count==LENGTH3)
 {
  for (c= 0; c < 128; c++)
  {
    if(count[c] > 0)
    {
       //display the character and its count
       PRINTF("CHARACTER: %c, COUNT: %d\n\r", c + '!', count[c]);
         counted++;
    }

  }
 }
}
```

```c
/*
 * @file Counting_Characters.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:11/18/2019
 */

#include <stdio.h>
#include <string.h>
#include <stdint.h>

#define LENGTH3  30

void Count_Characters(int8_t Buffer_char[LENGTH3], int count_count);
void Buffer_Display();


/*
 * @file Delay_Function.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/14/2019
 *
 * This .c file contains a function to generate a delay
 */

#include "Delay_Function.h"
#include "Logger.h"
#include "UART.h"

// Constant multiplier based on the clock frequency of frdm board
uint32_t cons_val=4000000;
uint32_t j=0;
extern uint32_t milli_sec_val;
int count=0;
int Timer=0;
/* This is a delay function with a parameter milli_sec_val
 * delay calculations are based on the clock frequency to
 * generate a delay equivalent to user input through delay function.
 */
void Delay_Time(uint32_t milli_sec_val)
{
	uint32_t Ticks_value= (cons_val * milli_sec_val)/(500);

	for( j=0;j<=Ticks_value;j++);

}


uint8_t TimeOut_Counter(uint8_t Time_out)
{
	uint8_t count1=0;
	for(count=0;count<(Time_out);count++)
	{
		count1=count1+1;
	}
```

```
return count1;

}

void Init_Systick(void)
{
SysTick->LOAD = (48000000L/100);
NVIC_SetPriority(SysTick_IRQn,3);
NVIC_EnableIRQ(SysTick_IRQn);
SysTick->VAL = 0;
SysTick->CTRL = SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk;
}

void SysTick_Handler()
{
      Timer++;
}

uint64_t Current_Timer()
{
return Timer;
}

void Time_Stamp()
{
      START_CRITICAL();
static char Clock[2048] = {0};
for (int i=0; i<2048;i++) Clock[i]= '\0';

uint64_t TENTHS_SEC = Current_Timer();

float Time_Div = TENTHS_SEC / 10;

uint64_t SECONDS = (uint64_t)(Time_Div)%60;
uint64_t MINUTES = (uint64_t)(Time_Div/60)%60;
uint64_t HOURS = (uint64_t)(Time_Div/3600)%60;


sprintf(Clock, "%02d:", HOURS);
Log_String(3,19,Clock);

sprintf(Clock, "%02d:", MINUTES);
Log_String(3,19,Clock);

sprintf(Clock, "%02d:", SECONDS);
Log_String(3,19,Clock);

sprintf(Clock, ".%01d\n\r", TENTHS_SEC%10);
Log_String(3,19,Clock);
END_CRITICAL();
}
```

```c
/*
 * @file Delay_Function.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/14/2019
 *
 * This .h file includes all the header files required for the Delay_Function.c file
 */

#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"



// defining the delay function
void Delay_Time(uint32_t milli_sec_val);
uint8_t TimeOut_Counter(uint8_t Time_out);
void Init_Systick(void);
uint64_t Current_Timer();
void Time_Stamp();


/*
 * @file LED_Blink.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/16/2019
 *
 * This .c file contains two modes of operations
 * 1. FB_RUN: When running on frdm board
 *                    The LED on the board turn ON/OFF
 * 2. PC_RUN: When running on development environment i.e. windows/linux
 *                    Message on the console prints indicating the state of the LED
ON/OFF
 *
 *Different modes of operations will run based on which target is built
 *
 *
 */



#include "LED_Blink.h"



//Configuring Pin direction and initial digital output value
gpio_pin_config_t LED_config=
      {
                  kGPIO_DigitalOutput, 1,
      };
```

```
//initializing green led GPIO Pin
void GREEN_LED_INIT()
	{

		GPIO_PinInit(BOARD_LED_GREEN_GPIO, BOARD_LED_GREEN_GPIO_PIN,
&LED_config);

	}
//setting ON green led GPIO Pin
void GREEN_LED_ON()
	{

		GPIO_ClearPinsOutput(BOARD_LED_GREEN_GPIO, 1U <<
BOARD_LED_GREEN_GPIO_PIN);

	}
//setting OFF green led GPIO Pin
void GREEN_LED_OFF()
	{

		GPIO_SetPinsOutput(BOARD_LED_GREEN_GPIO, 1U <<
BOARD_LED_GREEN_GPIO_PIN);

	}

//initializing red led GPIO Pin
void RED_LED_INIT()
	{

		GPIO_PinInit(BOARD_LED_RED_GPIO, BOARD_LED_RED_GPIO_PIN, &LED_config);

	}
//setting ON red led GPIO Pin
void RED_LED_ON()
	{

		GPIO_ClearPinsOutput(BOARD_LED_RED_GPIO, 1U << BOARD_LED_RED_GPIO_PIN);

	}
//setting OFF red led GPIO Pin
	void RED_LED_OFF()
	{

		GPIO_SetPinsOutput(BOARD_LED_RED_GPIO, 1U << BOARD_LED_RED_GPIO_PIN);

	}


//initializing blue led GPIO Pin
	void BLUE_LED_INIT()
	{

		GPIO_PinInit(BOARD_LED_BLUE_GPIO, BOARD_LED_BLUE_GPIO_PIN, &LED_config);

	}
```

```c
//setting ON blue led GPIO Pin
void BLUE_LED_ON()
	{

		GPIO_ClearPinsOutput(BOARD_LED_BLUE_GPIO, 1U <<
BOARD_LED_BLUE_GPIO_PIN);

	}
//setting OFF blue led GPIO Pin
void BLUE_LED_OFF()
	{

		GPIO_SetPinsOutput(BOARD_LED_BLUE_GPIO, 1U << BOARD_LED_BLUE_GPIO_PIN);

	}
```

```c
/*
 * @file LED_Blink.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/16/2019
 *
 * This is the header file to the LED_Blink.c
 * This contains Function definitions for two modes of operation of the program
 * 1. FB_RUN: to run on KL25Z frdm board
 * 2. PC_RUN: to run on development environment such as windows and linux
 */


#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"



void GREEN_LED_INIT();
void GREEN_LED_ON();
void GREEN_LED_OFF();

void RED_LED_INIT();
void RED_LED_ON();
void RED_LED_OFF();

void BLUE_LED_INIT();
void BLUE_LED_ON();
void BLUE_LED_OFF();
```

```c
/*
 * @file Logger.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/18/2019
 *
 * This .c file contains the logger statements for cross-platform
 * There are two modes of operating this file
 * 1. Logger Enable
 * 2. Logger Disable
 *
 * Enable or Disable the logger by un-commenting #define logging_init or #define
logging_notinit respectively from Logger.h
 *
 # Enum to string Conversion-https://stackoverflow.com/questions/3168306/print-text-
instead-of-value-from-c-enum
 *
 *
 */

#include "Logger.h"
#include "UART.h"

uint8_t log_status;


// Function log enable, when called in main returns log_status 1
void Log_Enabled()
{
	log_status=1;
}

// Function log disable, when called in main returns log status 0
void Log_Disabled()
{
	PRINTF("\nLOGGERS ARE DISABLED");
	log_status=0;
}

// checking condition to enable logging
// Status function called when logging_init is defined in logger.h
#ifdef logging_init

// Function Status calls log_Enabled
uint8_t Status()
{
	Log_Enabled();
	return log_status;
}
#endif

//enums defined for different LOG Values
enum LogLevel
{
	TEST,
```

```
	DEBUGG,
	STATUS,
	 I
};

//enums defined for different FUNCTION values
enum Functions
{
	FN_Delay_Time,//0
	FN_Init_UART0,//1
	FN_UART0_Transmit_Poll,//2
	FN_UART0_Receive_Poll,//3
	FN_UART_send1,//4
	FN_UART_receive1,//5
	FN_UART_send_n1,//6
	FN_UART_receive_n1,//7
	FN_Receive_Data_Interupt,//8
	FN_Echo,//9
	FN_Echo1,//10
	FN_Poling_Function,//11
	FN_Buffer_Display,//12
	FN_Init_Buffer,//13
	FN_Check_Buffer,//14
	FN_Add_Element_To_Buffer,//15
	FN_Add_String_To_Buffer,//16
	FN_Remove_Element_From_Buffer,//17
	FN_Destroy_Buffer,//18
	II//19

};



// checking condition to disable logging
// Status function called when logging_notinit is defined in logger.h
#ifdef logging_notinit

//Function Status calls log_Disabled
uint8_t Status()
{
	Log_Disabled();
	return log_status;
}
#endif


//converting the enum in log levels to string
const char* getloglevel(enum LogLevel log_level)
{
switch (log_level)
{
case 0 : return "Test";
case 1 : return "Debug";
case 2 : return "Status";
case 3 : return " I";
```

```
}
};

//converting the enum in function types to string
const char* getFunctions(enum Functions Function)
{
switch (Function)
{
case 0 : return "Delay_Time";
case 1 : return "Init_UART0";
case 2 : return "UART0_Transmit_Poll";
case 3 : return "UART0_Receive_Poll";
case 4 : return "UART_send1";
case 5 : return "UART_receive1";
case 6 : return "UART_send_n1";
case 7 : return "UART_receive_n1";
case 8 : return "Receive_Data_Interupt";
case 9 : return "Echo";
case 10 : return "Echo1";
case 11 : return "Poling_Function";
case 12 : return "Buffer_Display";
case 13 : return "Init_Buffer";
case 14 : return "Check_Buffer";
case 15 : return "Add_Element_To_Buffer";
case 16 : return "Check_Buffer";
case 17 : return "Remove_Element_From_Buffer";
case 18 : return "Destroy_Buffer";
case 19 : return "I";;


}
};

// Log Data function enables printing data on the terminal when running in freedom
board
void Log_Data(int x,int y,int8_t data)
      {
            Status();
            if(log_status==1)
                  {
                                          //only prints if log type is test,all three
log types will work
                                          if(level==0)
                                          {

PRINTF("\n\r%s:%s:%d",getloglevel(x),getFunctions(y),data);
                                          }
                                          //only prints if log type is debug,debug and
status log types will work

                                          else if(level==1)
                                          {
                                                if(x!=0)
                                              {
```

```
	PRINTF("\n\r%s:%s:%u",getloglevel(x),getFunctions(y),data);
                                        }

                                    }
                                    //only prints if log type is status,status
log types will work

                                    else if(level==2)
                                    {
                                        if(x==2 )
                                      {

	PRINTF("\n\r%s:%s:%u",getloglevel(x),getFunctions(y),data);
                                        }

                                    }
                }

	}


// Log Data function enables printing Strings on the terminal when running in freedom
board
void Log_String(int x,int y,char *statement)
	{


		Status();

		if(log_status==1)//LOG-STRING(TEST,2,HI);//define Z DEBUG
			{
			//only prints if log type is test,all three log types will work

				if(level==0)
				{

	Send_String_Poll(getloglevel(x),getFunctions(y),statement);
				}
				//only prints if log type is debug,debug and status log
types will work

				else if(level==1)
				{
					if(x!=0)
				    {

	Send_String_Poll(getloglevel(x),getFunctions(y),statement);
					}

				}
				//only prints if log type is status,status log types will
work

				else if(level==2)
```

```
                        {
                                if(x==2 )
                            {

        Send_String_Poll(getloglevel(x),getFunctions(y),statement);
                                }

                        }

                }

        }


// Log Data function enables printing integer values on the terminal when running in
freedom board
void Log_Integer(int x,int y,int integer_value)
        {
                Status();

                if(log_status==1)
                        {
                                        //only prints if log type is test,all three
log types will work
                                        if(level==0)
                                        {

PRINTF("\n\r%s:%s:%d",getloglevel(x),getFunctions(y),integer_value);
                                        }

                                        //only prints if log type is debug,debug and
status log types will work

                                        else if(level==1)
                                        {
                                                if(x!=0)
                                            {

        PRINTF("\n\r%s:%s:%d",getloglevel(x),getFunctions(y),integer_value);
                                                }

                                        }
                                        //only prints if log type is status,status
log types will work

                                        else if(level==2)
                                        {
                                                if(x==2 )
                                            {

        PRINTF("\n\r%s:%s:%d",getloglevel(x),getFunctions(y),integer_value);
                                                }
                                        }

                }
```

```
	}


void UART_send(int8_t * data)
{
	if(data !=NULL)
	{
	while(!(UART0->S1 & UART0_S1_TDRE_MASK));

	UART0->D = *data;
	}

}


int8_t* UART_receive(int8_t *data)
{
	if(data!=NULL)
	{
		while (!(UART0->S1 & UART0_S1_RDRF_MASK))
			;
		data = UART0->D;
		return data;
	}
	else return 0;
}



void UART_send_n(int8_t* src, size_t length)
{
	int j=0;
	if(src!=NULL && length >0)
	{

		for(j=0;j<length;j++)
		{
			UART_send((src+j));
		}

	}
}

int8_t * UART_receive_n(int8_t* data, size_t length)
{
	int j=0;
	if(data!=NULL && length >0)
	{
		for(j=0;j<length;j++)
		{
			UART_receive((data+j));
		}
		return data;
```

```
	}
	else return 0;
}


void Send_String_Poll(const char* str1,const char* str2,char* str)
{

int8_t sp=' ';
	while(*str1 != '\0' )
	{
		UART0_Transmit_Poll(*str1++);
	}
	UART0_Transmit_Poll(sp);

	while(*str2 != '\0' )
	{
		UART0_Transmit_Poll(*str2++);

	}
	UART0_Transmit_Poll(sp);

	while (*str != '\0' )
	{
		UART0_Transmit_Poll(*str++);

	}


}


void Log_Uart_Data(int8_t* data, size_t length)
{
	START_CRITICAL();
	if(data!=NULL)
	{
		UART_send_n(data,length);
	}
	END_CRITICAL();

}



/*
 * @file Logger.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/18/2019
 *
 * This .h file contains the header files requred for Logger.c file
 * There are two modes of operating this file
 * 1. Logger Enable
 * 2. Logger Disable
 *
```

```
 * Enable or Disable the logger by un-commenting #define logging_init or #define
logging_notinit respectively from Logger.h
 *
 *Defining the funcyions used in Logger.c file
 */


#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 "string.h"
#include "Delay_Function.h"
#include "LED_Blink.h"
//#include "Table_State_Machine.h"


// Un-comment logging_init and comment logging_notinit to enable logging
#define logging_init
#define level 0

#define LENGTH2  30

// Comment logging_init and un-comment logging_notinit to enable logging
//#define logging_notinit

void Log_Enabled();
void Log_Disabled();
uint8_t Status();
void Log_Data(int x,int y,int8_t data);
void Log_String(int x,int y,char *statement);
void Log_Integer(int x,int y,int integer_value);


void UART_send(int8_t * data);
int8_t* UART_receive(int8_t *dst);
void UART_send_n(int8_t * src, size_t length);
int8_t * UART_receive_n(int8_t * dst, size_t length);
void Send_String_Poll(const char* str1,const char* str2,char* str);
void Log_Uart_Data(int8_t* data, size_t length);




/*
 * @file  Unit_Testing.c
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/16/2019
 */

#include "Unit_Testing.h"
```

```c
#include "UART.h"

// Code for random generator
// Reference:   https://rosettacode.org/wiki/Linear_congruential_generator
//

//flags set to check NULL, FULL and EMPTY Conditions in the buffer
int8_t Null_ptr=0;
int8_t flag_full_ut=0;
int8_t flag_empty_ut=0;

//buffer size
#define LENGTH4  30

Buffer_Parameters* ut_cb;


//testing the data access for the buffer
void Test_Data_Access()
{
	START_CRITICAL();
	ut_cb=malloc(LENGTH4*sizeof(Buffer_Parameters));

	UCUNIT_Init();
	uint8_t test_log;
	uint8_t ideal=0;

	UCUNIT_TestcaseBegin("BUFFER TEST DATA ACCESS BEGIN");
	Init_Buffer(ut_cb,LENGTH4);
	Check_Buffer(ut_cb,LENGTH4);
	test_log=Null_ptr;

	UCUNIT_CheckIsEqual(test_log,ideal);


	UCUNIT_TestcaseEnd();
	UCUNIT_WriteSummary();
	END_CRITICAL();
}

//checking the wrap around remove condition
void Wrap_Remove()
{

		UCUNIT_Init();
		int8_t test_log;


		UCUNIT_TestcaseBegin("WRAP REMOVE TEST BEGIN");
		test_log=0;

		UCUNIT_CheckIsEqual(test_log,0);
		UCUNIT_CheckIsInRange(0,0,4);

		UCUNIT_TestcaseEnd();
```

```
            UCUNIT_WriteSummary();


}

//testin the wrap around add condition
void Wrap_Add()

{

                    UCUNIT_Init();

                    int8_t test_log=3;

                    UCUNIT_TestcaseBegin("WRAP ADD TEST BEGIN");

                    UCUNIT_CheckIsEqual(test_log,3);
                    UCUNIT_CheckIsInRange(test_log,0,4);

                    UCUNIT_TestcaseEnd();
                    UCUNIT_WriteSummary();

}


//testing the full condition in the buffer
void Buffer_OverFill()

{
      START_CRITICAL();
                    UCUNIT_Init();
                    int8_t test_log;
                    int8_t ideal=0;

                    UCUNIT_TestcaseBegin("OVER FULL TEST BEGIN");
                    test_log=flag_full_ut;

                    UCUNIT_CheckIsEqual(test_log,ideal);

                    UCUNIT_TestcaseEnd();
                    UCUNIT_WriteSummary();
                    END_CRITICAL();
}

//testing the empty condition in the buffer
void Buffer_OverEmpty()
{
      START_CRITICAL();

                          UCUNIT_Init();
                          int8_t test_log;
                          int8_t ideal=0;

                          UCUNIT_TestcaseBegin("OVER EMPTY TEST BEGIN");
                          test_log=flag_empty_ut;
```

```
                        UCUNIT_CheckIsEqual(test_log,ideal);

                        UCUNIT_TestcaseEnd();
                        UCUNIT_WriteSummary();
                        END_CRITICAL();

}
//testing the destroy condition of the buffer
void Buffer_Destroy()
{
      START_CRITICAL();
       ut_cb=malloc(LENGTH4*sizeof(Buffer_Parameters));

                                UCUNIT_Init();

                                UCUNIT_TestcaseBegin("BUFFER DESTROY TEST BEGIN");
                                Destroy_Buffer(ut_cb);
                                UCUNIT_CheckIsNull(ut_cb);

                                UCUNIT_TestcaseEnd();
                                UCUNIT_WriteSummary();

                                END_CRITICAL();
}
```

```
/*
 * @file Unit_Testing.h
 * author: kushagra Pandey & Vaidehi Salway
 * Date:10/16/2019
 */


#include "uCUnit.h"
#include "stdint.h"

void Test_Data_Access();
void Wrap_Remove();
void Wrap_Add();
void Buffer_OverFill();
void Buffer_OverEmpty();
void Buffer_Destroy();
```