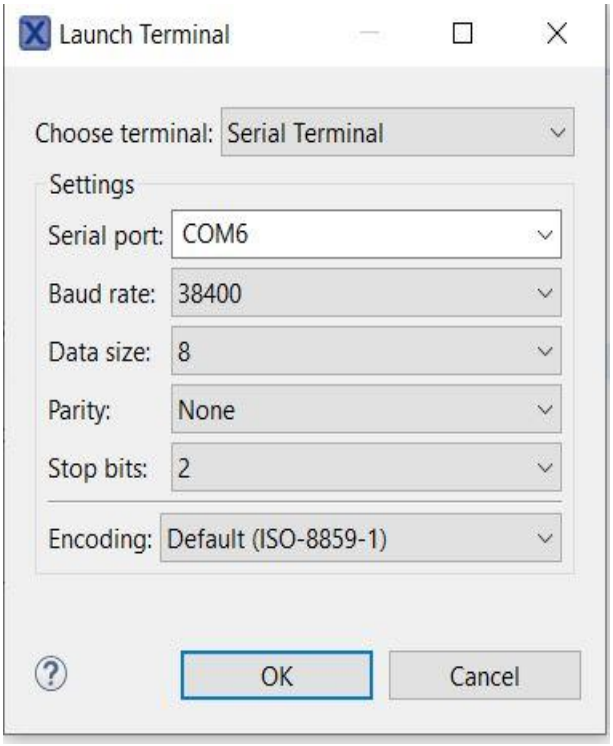
PES Assignment – 6

Github : https://github.com/kevintom98/PES-Assignment-6

Screenshots

```
Welcome to BreakfastSerial!
? author
Kevin Tom
? Author
Kevin Tom
? DUMP 0 64
0000_0000  00 30 00 20 D5 00 00 00 43 01 00 00 F9 12 00 00
0000_0010  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0000_0020  00 00 00 00 00 00 00 00 00 00 00 00 47 01 00 00
0000_0030  00 00 00 00 00 00 00 00 49 01 00 00 4B 01 00 00
? dump a0 0x20
0000_00a0  0F 02 00 00 17 02 00 00 1F 02 00 00 27 02 00 00
0000_00b0  2F 02 00 00 37 02 00 00 3F 02 00 00 47 02 00 00
? print
Unknown Command: print
? help
author - This command will print the name of the author who wrote the command line
dump - This command will print Hexdump of memory(eg: dump start_addr end_addr ; dump 0 0x64)
? prin
```



```
/*
 *    cbfifo.c - Circular buffer implementation
 *
 *    Author: Kevin Tom, keto9919@colorado.edu
 */


#include <stdint.h>
#include <stdio.h>
#include <stdbool.h>
#include ”cbfifo.h”
#include <MKL25Z4.h>


#define MAX_SIZE 256 //Max size of the circular buffer is 128 bytes


//Structure for the circular buffer
typedef struct cbfifo_struct
{
      uint16_t buf[MAX_SIZE];
      uint16_t head;
      uint16_t tail;
      uint16_t length;
      uint16_t full;
}cbfifo_t;


//Object of the structure
```

```
cbfifo_t q[INSTANCES];



/*
 * This function enqueue data into the buffer, up to 128 bytes.
 *
 * Parameters :
 *   buf - pointer to the data to be enqueued
 *   nbyte - number of bytes to be enqueued
 *
 * Returns :
 *   Number of bytes enqueued (size_t - variable).
 *   In case of error -1.
 *
 */
size_t cbfifo_enqueue(void *buf, size_t nbyte, inst ins)
{
     uint8_t a;
     size_t temp;
     uint32_t masking_state;


     // save current masking state
     masking_state = __get_PRIMASK();
     // disable interrupts
     __disable_irq();


     /**********************INITIAL
CONDITIONS*****************************/
     //Case when nbyte is 0
     if(nbyte == 0)
```

```
	{
		__set_PRIMASK(masking_state);
		return 0;
	}

	//Case when *buf is NULL
	if(buf == NULL)
	{
		__set_PRIMASK(masking_state);
		return -1;
	}

	//When nbyte>128 truncating it to 128
	if(nbyte > MAX_SIZE)
		nbyte = MAX_SIZE;

	//If buffer is full returning 0
	if(q[ins].full == 1)
	{
		__set_PRIMASK(masking_state);
		return 0;
	}

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

	temp = q[ins].length; //temp for calculating length difference

	if((q[ins].head == MAX_SIZE)) //Wrapping head around
		q[ins].head=0;


	for(int i = 0;i<nbyte;i++)
```

```
      {
            if(q[ins].full == 1) // Checking if full
                  break;

            a = *(uint8_t *)(buf+i);           //  converting  data  in
pointer location to uint_8
          q[ins].buf[q[ins].head]=a;        //Writing data into buffer

            (q[ins].head)++;  //Updating head
            (q[ins].length)++; //updating length

            if((q[ins].head == MAX_SIZE)) //Wrapping head around
                  q[ins].head=0;
            if(q[ins].head == q[ins].tail)
                  q[ins].full = 1;
      }

      __set_PRIMASK(masking_state);
      return (q[ins].length - temp);
}
```

```
/*
 * This function dequeue data into the buffer, up to 128 bytes.
 *
 * Parameters :
 *   buf - pointer to the destination
 *   nbyte - number of bytes to be dequeued
 *
 * Returns :
 *   Number of bytes dequeued (size_t - variable).
```

```
 *    In case of error -1.
 *
 */
size_t cbfifo_dequeue(void *buf, size_t nbyte, inst ins)
{
      uint16_t temp;
      temp = q[ins].length;

      uint32_t masking_state;

      // save current masking state
      masking_state = __get_PRIMASK();
      // disable interrupts
      __disable_irq();




      /************************INITIAL
CONDITIONS*****************************/
      // If buf == NULL error
      if(buf == NULL)
      {
            __set_PRIMASK(masking_state);
            return -1;
      }

      //Case when nbyte is 0
      if(nbyte == 0)
      {
            __set_PRIMASK(masking_state);
            return 0;
      }
```

```
	//When nbyte>128 truncating it to 128
	if(nbyte > MAX_SIZE)
		nbyte = MAX_SIZE;

	// Buffer is empty
	if((q[ins].head==q[ins].tail) && (q[ins].full == 0))
	{
		__set_PRIMASK(masking_state);
		return 0;
	}

	//truncating nbyte to available elements
	if(nbyte > q[ins].length)
		nbyte = q[ins].length;

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



	if(q[ins].length != 0)
	{
		/*Case where nbyte is greater than length*/
		if(nbyte > q[ins].length)
		{
			for(int i=0;i<q[ins].length;i++)
			{
				//Checking if buffer is empty
				if((q[ins].head==q[ins].tail) && (q[ins].full ==
0))
				{
					break;
				}
```

```
                    *((uint8_t *)buf+i) = q[ins].buf[q[ins].tail];

                    (q[ins].tail)++;                //updating   tail
(incrementing)
                    (q[ins].length)--;            //updating    length
(decrementing)

                    if(q[ins].full == 1) //if a dequeue happens full
flag should be set to 0
                         q[ins].full=0;
                    if(q[ins].tail  ==  MAX_SIZE)    //tail  wrapping
around
                         q[ins].tail = 0;
               }
               __set_PRIMASK(masking_state);
               return   (temp   -   q[ins].length);   //returning   the
difference in lengths
          }

          for( int i=0;i<nbyte;i++)
          {
               //Checking if buffer is empty
               if((q[ins].head==q[ins].tail) && (q[ins].full == 0))
                    break;

               *((uint8_t *)buf+i) = q[ins].buf[q[ins].tail];

               (q[ins].tail)++;
               (q[ins].length)--;

               //if a dequeue happens full flag should be set to 0
               if(q[ins].full == 1)
                    q[ins].full=0;
               //tail wrapping around
               if(q[ins].tail == MAX_SIZE)
```

```
                        q[ins].tail = 0;
            }
            __set_PRIMASK(masking_state);
            //returning the difference in lengths
            return (temp - q[ins].length);
      }

      __set_PRIMASK(masking_state);
      return 0; //if control reaches here return 0
}
```

```
/*
 *This function returns the length of the buffer
 *
 * Parameter :
 *    None
 * Return :
 *    length of the buffer (size_t)
 */
size_t cbfifo_length(inst ins)
{
      return q[ins].length;
}
```

```
/*
 *This function returns the capacity of the buffer
```

```
 *
 * Parameter :
 *    None
 * Return :
 *    capacity of the buffer (size_t)
 */
size_t cbfifo_capacity(inst ins)
{
      return MAX_SIZE;
}


/* This function dumps characters in the buffer as
 *  char.
 * Parameter:
 *    None
 * Return:
 *    None
 *
 */
void cbfifo_dump(inst ins)
{
      for(int i =0;i<MAX_SIZE;i++)
            printf("%c",q[ins].buf[i]); //This is used for dumping data
}

/*
 * cbfifo.h - a fixed-size FIFO implemented via a circular buffer
 *
 * Author: Howdy Pierce, howdy.pierce@colorado.edu
 *
 */
```

```
#ifndef _CBFIFO_H_
#define _CBFIFO_H_

#include <stdlib.h>  // for size_t



//Number of CBFIFO instances in use
#define INSTANCES  (2)

//Name of the instances
typedef enum instance
{
     RX_Buffer,
     TX_Buffer,
}inst;




/*
 * Enqueues data onto the FIFO, up to the limit of the available FIFO
 * capacity.
 *
 * Parameters:
 *   buf      Pointer to the data
 *   nbyte    Max number of bytes to enqueue
 *
 * Returns:
 *   The number of bytes actually enqueued, which could be 0. In case
 * of an error, returns -1.
 */
size_t cbfifo_enqueue(void *buf, size_t nbyte, inst ins);
```

```
/*
 * Attempts to remove (”dequeue”) up to nbyte bytes of data from the
 * FIFO. Removed data will be copied into the buffer pointed to by buf.
 *
 * Parameters:
 *   buf      Destination for the dequeued data
 *   nbyte    Bytes of data requested
 *
 * Returns:
 *   The number of bytes actually copied, which will be between 0 and
 * nbyte.
 *
 * To further explain the behavior: If the FIFO’s current length is 24
 * bytes, and the caller requests 30 bytes, cbfifo_dequeue should
 * return the 24 bytes it has, and the new FIFO length will be 0. If
 * the FIFO is empty (current length is 0 bytes), a request to dequeue
 * any number of bytes will result in a return of 0 from
 * cbfifo_dequeue.
 */
size_t cbfifo_dequeue(void *buf, size_t nbyte, inst ins);


/*
 * Returns the number of bytes currently on the FIFO.
 *
 * Parameters:
 *   none
 *
 * Returns:
 *   Number of bytes currently available to be dequeued from the FIFO
 */
```

```
size_t cbfifo_length(inst ins);


/*
 * Returns the FIFO’s capacity
 *
 * Parameters:
 *   none
 *
 * Returns:
 *   The capacity, in bytes, for the FIFO
 */
size_t cbfifo_capacity(inst ins);


/* This function dumps characters in the buffer as
 *  char.
 * Parameter:
 *   None
 * Return:
 *   None
 *
 */
void cbfifo_dump(inst ins);


#endif // _CBFIFO_H_


/*
 * command_processor.c
 *
 *  Created on: 08-Nov-2021
```

```
 *      Author: Kevin Tom, Kevin.Tom@colorado.edu
 *
 *
 *  This file has the function implementation of command processor
 */

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

#include ”command_processor.h”
#include ”hexdump.h”




/* This function is the handler for author command.
 * This function prints the authors name
 *
 * Parameters:
 * None
 *
 * Returns:
 * None
 *
 *
 * */
void author_handler()
{
     printf(”\n\rKevin Tom\n\r”);
```

```
}
```

```
/* This function is the handler for author command.
 * This function prints the authors name
 *
 * Parameters:
 *    argc - Number of arguments
 *    argv - Array of arguments ending with '\0'
 *
 * Returns:
 * None
 *
 *
 * */
void dump_handler(int argc, char *argv[])
{

      int start = 0, len = 0;

      //Converting start address to from hex to decimal
      start = (int)strtol(argv[1],NULL,16);

      /*If given address is in hex convert it into decimal
       * Else convert it into decimal from string
       */
      if((*(argv[2]) == '0') && (*(argv[2]+1) == 'x'))
            len = (int)strtol(argv[2],NULL,16);
```

```
      else
            len = atoi(argv[2]);


      //Calling the hexdump function
      hexdump((int *)start,len);
}




/* This function is the handler for help command
 * it prints the help menu
 *
 * Parameters:
 *    argc - Number of arguments
 *    argv - Array of arguments ending with '\0'
 *
 * Returns:
 *    None
 * */
void help_handler()
{
      for (int i=0; i < num_commands; i++)
        {
            //Prints until help
          if (strcasecmp("help", commands[i].name) != 0)
          {
            //Printing the string
            for(const char *j=commands[i].help_string; *j != '\0'; j++)
                  printf("%c",*j);
          }
        }
```

```
}
```

```
/* This function starts the command processor and handles the commands
recevied
 *
 * Parameters:
 *    None
 *
 * Returns:
 * None
 *
 *
 * */
void command_processor_start()
{
      char command[100];
      int i=-1;
      printf("\n\n\rWelcome to BreakfastSerial!\n\r");

      while(1)
      {
            printf("? ");
            i= -1;

            /**********Accumulator***********/
            while(command[i] != '\r')
            {
                  i++;
                  //Getting character
```

```
                command[i]= getchar();

                //Handling backspace
                if((command[i] == '\b') && (i > 1))
                {
                      command[i] =' ';
                      printf(" \b \b");
                }

          }
          command[i++] = '\0';
          /*****************************/

          //Calling process command function
          process_command(command);
     }


}
```

```
/* This function splits the received command into
 * argc and argv vectors and calls appropriate handling functions
 *
 * Parameters:
 *    char *input - Input string from accumualtor
 *
 * Returns:
 *    None
 *
```

```
 *
 * */
void process_command(char *input)
{
  char *p = input;
  char *end;


  // find end of string
  for (end = input; *end != '\0'; end++)
    ;


  //Bool for printing error message
  bool found = false;
  char *argv[10];
  int argc = 0;


  __builtin_memset(argv, 0, sizeof(argv));


  for (p = input; p < end; p++)
  {
       //If a character is recognized
       if((*p >= 48))
       {
            //if previous character is ' ' or '\0' or it is starting
character
            if( (*(p-1) ==' ') || (p == input) || (*(p-1) =='\0'))
            {
                 //Write the address to argv[argc]
                 argv[argc] = p;
```

```
                //Incrementing argc
                argc++;
            }
            //If trailing character is space make it as '\0'
            if(*(p+1) == ' ')
                    *(p+1) = '\0';
      }
}


//If no command received
argv[argc] = NULL;
if (argc == 0)   // no command
  return;


//Checking which handler to call using argv[0] string
for (int i=0; i < num_commands; i++)
{
  if (strcasecmp(argv[0], commands[i].name) == 0)
  {
    commands[i].handler(argc, argv);
    found = true;
    break;
  }
}

//If no handler is found print error message
if(found == false)
{
      printf("\n\rUnknown Command: ");
      for(char *i= argv[0];*i != '\0' ;i++)
            printf("%c",*i);
```

```
        printf("\n\r");
    }
}

/*
 * command_processor.h
 *
 *  Created on: 04-Nov-2021
 *      Author: Kevin Tom, Kevin.Tom@colorado.edu
 */

#ifndef _COMMAND_PROCESSOR_H_
#define _COMMAND_PROCESSOR_H_


//Function pointer for each function
typedef void (*command_handler_t)(int, char *argv[]);


//Structure which holds all the handler details
typedef struct
{
  const char *name;
  command_handler_t handler;
  const char *help_string;
} command_table_t;


/* This function is the handler for author command.
 * This function prints the authors name
 *
 * Parameters:
 * None
```

```
 *
 * Returns:
 * None
 *
 *
 * */
void author_handler();



/* This function is the handler for author command.
 * This function prints the authors name
 *
 * Parameters:
 *    argc - Number of arguments
 *    argv - Array of arguments ending with '\0'
 *
 * Returns:
 * None
 * */
void dump_handler(int argc, char *argv[]);



/* This function is the handler for help command
 * it prints the help menu
 *
 * Parameters:
 *    argc - Number of arguments
 *    argv - Array of arguments ending with '\0'
 *
 * Returns:
```

```
 *    None
 * */
void help_handler();




/*************************************************COMMAND
TABLE*************************************************************
***/
static const command_table_t commands[] =
{
           {"author", author_handler,"\n\rauthor - This command will
print the name of the author who wrote the command line\n\r"},
           {"dump", dump_handler,"dump - This command will print Hexdump
of memory(eg: dump start_addr end_addr ; dump 0 0x64)\n\r"},
           {"help", help_handler," "}
};
/*************************************************************
*************************************************************/




//Calculating number of commands
static    const    int    num_commands    =    sizeof(commands)    /
sizeof(command_table_t);




/* This function starts the command processor and handles the commands
recevied
 *
 * Parameters:
 *    None
 *
 * Returns:
 * None
```

```
 * */
void command_processor_start();



/* This function splits the received command into
 * argc and argv vectors and calls appropriate handling functions
 *
 * Parameters:
 *    char *input - Input string from accumualtor
 *
 * Returns:
 *    None
 *
 *
 * */
void process_command(char *input);


#endif // _COMMAND_PROCESSOR_H_
/*
 * hexdump.c
 *
 *  Created on: 09-Nov-2021
 *      Author: Kevin Tom, Kevin.Tom@colorado.edu
 *
 *
 *      This file has the hexdump function
 */


#include ”hexdump.h”
```

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




/* This converts integer input to hexadecimal character
 *
 *
 * Parameters:
 *         uint32_t x - Number to be converted
 *
 * Returns:
 *         char       - Hexadecimal equivalent
 * */
char int_to_hexchar(uint32_t x)
{
     if (x < 10)
          return ('0' + x);
     else
          return ('A' + (x - 10));
}




/* This function prints the hexdump starting from an address till the
 * given length.
 *
```

```
 *
 * Parameters:
 *         int *start - Start address
 *         size_t len - Total number of locations to be printed
 *
 * Returns:
 *         None
 * */
void hexdump(int *start, size_t len)
{
      uint8_t *buf = (uint8_t*) start;
      int start_addr = (int)start;


      //Truncating i the len is greater than MAX Size (640)
      if(len > DUMP_MAX_SIZE)
      {
            len = DUMP_MAX_SIZE;
      }

      //Printing in a new line
      printf("\n\r");

      for(int i =0;i<len;i+=STRIDE)
      {
            //Printing the address
            printf("%04x_%04x",(start_addr & (0xFFFF0000)),(start_addr &
(0x0000FFFF)));
          printf("  ");

          //Printing the memory content
          for (int j=0; (j < STRIDE) && (i+j < len); j++)
            {
```

```
            printf("%c",int_to_hexchar((buf[i+j]) >> 4));
                  printf("%c",int_to_hexchar((buf[i+j]) & 0x0f));
                  printf(" ");
            }

          //Incrementing the address by STRIDE
            start_addr += STRIDE;

            //Going to new line
                  printf("\r");
                  printf("\n");
      }
}
```

```
/*
 * hexdump.h
 *
 *  Created on: 09-Nov-2021
 *      Author: Kevin Tom, Kevin.Tom@colorado.edu
 */


#ifndef _HEXDUMP_H_
#define _HEXDUMP_H_


#include <stddef.h>
#include <stdint.h>
```

```
#define STRIDE           (16)
#define DUMP_MAX_SIZE    (640)
```

```
/* This converts integer input to hexadecimal character
 *
 *
 * Parameters:
 *         uint32_t x - Number to be converted
 *
 * Returns:
 *         char       - Hexadecimal equivalent
 * */
char int_to_hexchar(uint32_t x);
```

```
/* This function prints the hexdump starting from an address till the
 * given length.
 *
 *
 * Parameters:
 *         int *start - Start address
 *         size_t len - Total number of locations to be printed
 *
 * Returns:
 *         None
 * */
void hexdump(int *start, size_t len);
```

```
#endif //_HEXDUMP_H_
```

```
/*
 * main.c - application entry point
 *
 * Author Howdy Pierce, howdy.pierce@colorado.edu
 */

#include ”sysclock.h”
#include ”UART.h”
#include ”test_cbfifo.h”
#include ”command_processor.h”


//Baud rate setting
#define BAUD_RATE 38400


int main(void)
{

  //Calling the system clock initialization function
  sysclock_init();

  //Calling the CBFIFO testing function
  test_cbfifo();
```

```
  //Calling the UART initialization function with BAUD_RATE
  Init_UART0(BAUD_RATE);


  //Starting the command processor
  command_processor_start();

  return 0 ;
}

/*
 * sysclock.c - configuration routines for KL25Z system clock
 *
 * Author Howdy Pierce, howdy.pierce@colorado.edu
 *
 * See section 24 of the KL25Z Reference Manual to understand this code
 *
 * Inspired by https://learningmicro.wordpress.com/configuring-device-
clock-and-using-systick-system-tick-timer-module-to-generate-software-
timings/

 */

#include "MKL25Z4.h"
#include "sysclock.h"


void sysclock_init()
{
  // Corresponds to FEI mode as shown in sec 24.4.1

  // Select PLL/FLL as clock source
```

```
  MCG->C1 &= ~(MCG_C1_CLKS_MASK);
  MCG->C1 |= MCG_C1_CLKS(0);

  // Use internal reference clock as source for the FLL
  MCG->C1 |= MCG_C1_IREFS(1);

  // Select the FLL (by setting "PLL select" to 0)
  MCG->C6 &= ~(MCG_C6_PLLS_MASK);
  MCG->C6 |= MCG_C6_PLLS(0);

  // Select 24 MHz - see table for MCG_C4[DMX32]
  MCG->C4 &= ~(MCG_C4_DRST_DRS_MASK & MCG_C4_DMX32_MASK);
  MCG->C4 |= MCG_C4_DRST_DRS(0);
  MCG->C4 |= MCG_C4_DMX32(1);
}



/*
 * sysclock.h - configuration routines for KL25Z system clock
 *
 * Author Howdy Pierce, howdy.pierce@colorado.edu
 */

#ifndef _SYSCLOCK_H_
#define _SYSCLOCK_H_

#define SYSCLOCK_FREQUENCY (24000000U)

/*
 * Initializes the system clock. You should call this first in your
 * program.
 */
void sysclock_init();
```

```
#endif  // _SYSCLOCK_H_

#include <stdio.h>
#include <stdint.h>
#include <assert.h>
#include <string.h>
#include ”cbfifo.h”



/*
 *This function TX_Buffers the cbfifo.c through different TX_Buffer cases.
 *Asserts are used for TX_Buffering. Program stops is there is a problem.
 *
 *Parameters:
 *    none
 *
 *return:
 *    none
 */
void test_cbfifo()
{
  char *str =”To be, or not to be: that is the question:\n”
    ”Whether ’tis nobler in the mind to suffer\n”
    ”The slings and arrows of outrageous fortune,\n”
    ”Or to take arms against a sea of troubles,\n”
    ”And by opposing end them? To die, to sleep--\n”
    ”No more--and by a sleep to say we end\n”
    ”The heart-ache and the thousand natural shocks\n”
    ”That flesh is heir to, ’tis a consummation\n”
    ”Devoutly to be wish’d. To die, to sleep;\n”
    ”To sleep: perchance to dream: ay, there’s the rub;\n”
```

```
    "For in that sleep of death what dreams may come\n"
    "When we have shuffled off this mortal coil,\n"
    "Must give us pause.";

  char buf[1024];
  const int cap = cbfifo_capacity(RX_Buffer);



  // asserts in following 2 lines -- this is not TX_Buffering the
student,
  // it's validating that the TX_Buffer is correct
  assert(strlen(str) >= cap*2);
  assert(sizeof(buf) > cap);
  assert(cap == 256 || cap == 127);



  assert(cbfifo_length(RX_Buffer) == 0);
  assert(cbfifo_dequeue(buf, cap,RX_Buffer) == 0);
  assert(cbfifo_dequeue(buf, 1,RX_Buffer) == 0);

  assert(cbfifo_length(TX_Buffer) == 0);
  assert(cbfifo_dequeue(buf, cap,TX_Buffer) == 0);
  assert(cbfifo_dequeue(buf, 1,TX_Buffer) == 0);

  // enqueue 10 bytes, then dequeue same amt
  assert(cbfifo_enqueue(str, 10,RX_Buffer) == 10);
  assert(cbfifo_length(RX_Buffer) == 10);
  assert(cbfifo_dequeue(buf, 10,RX_Buffer) == 10);
  assert(strncmp(buf, str, 10) == 0);
  assert(cbfifo_length(RX_Buffer) == 0);
```

```
 // enqueue 10 bytes, then dequeue same amt
assert(cbfifo_enqueue(str, 10,TX_Buffer) == 10);
assert(cbfifo_length(TX_Buffer) == 10);
assert(cbfifo_dequeue(buf, 10,TX_Buffer) == 10);
assert(strncmp(buf, str, 10) == 0);
assert(cbfifo_length(TX_Buffer) == 0);



// enqueue 20 bytes;  dequeue 5, then another 20
assert(cbfifo_enqueue(str, 20,RX_Buffer) == 20);
assert(cbfifo_length(RX_Buffer) == 20);

 // enqueue 20 bytes;  dequeue 5, then another 20
assert(cbfifo_enqueue(str, 20,TX_Buffer) == 20);
assert(cbfifo_length(TX_Buffer) == 20);



assert(cbfifo_dequeue(buf, 5, RX_Buffer) == 5);
assert(cbfifo_length(RX_Buffer) == 15);
assert(cbfifo_dequeue(buf+5, 20,RX_Buffer) == 15);
assert(cbfifo_length(RX_Buffer) == 0);
assert(strncmp(buf, str, 20) == 0);

assert(cbfifo_dequeue(buf, 5, TX_Buffer) == 5);
assert(cbfifo_length(TX_Buffer) == 15);
assert(cbfifo_dequeue(buf+5, 20,TX_Buffer) == 15);
assert(cbfifo_length(TX_Buffer) == 0);
assert(strncmp(buf, str, 20) == 0);

// fill buffer and then read it back out
```

```
assert(cbfifo_enqueue(str, cap, RX_Buffer) == cap);
assert(cbfifo_length(RX_Buffer) == cap);
assert(cbfifo_enqueue(str, 1,RX_Buffer) == 0);
assert(cbfifo_dequeue(buf, cap,RX_Buffer) == cap);
assert(cbfifo_length(RX_Buffer) == 0);
assert(strncmp(buf, str, cap) == 0);

// Add 20 bytes and pull out 18
assert(cbfifo_enqueue(str, 20,RX_Buffer) == 20);
assert(cbfifo_length(RX_Buffer) == 20);
assert(cbfifo_dequeue(buf, 18,RX_Buffer) == 18);
assert(cbfifo_length(RX_Buffer) == 2);
assert(strncmp(buf, str, 18) == 0);


// fill buffer and then read it back out
assert(cbfifo_enqueue(str, cap, TX_Buffer) == cap);
assert(cbfifo_length(TX_Buffer) == cap);
assert(cbfifo_enqueue(str, 1,TX_Buffer) == 0);
assert(cbfifo_dequeue(buf, cap,TX_Buffer) == cap);
assert(cbfifo_length(TX_Buffer) == 0);
assert(strncmp(buf, str, cap) == 0);

// Add 20 bytes and pull out 18
assert(cbfifo_enqueue(str, 20,TX_Buffer) == 20);
assert(cbfifo_length(TX_Buffer) == 20);
assert(cbfifo_dequeue(buf, 18,TX_Buffer) == 18);
assert(cbfifo_length(TX_Buffer) == 2);
assert(strncmp(buf, str, 18) == 0);

// Take out the 2 remaining bytes from above
assert(cbfifo_dequeue(buf, 2, RX_Buffer) == 2);
assert(strncmp(buf, str+18, 2) == 0);
```

```
// Take out the 2 remaining bytes from above
assert(cbfifo_dequeue(buf, 2, TX_Buffer) == 2);
assert(strncmp(buf, str+18, 2) == 0);


// write more than capacity
assert(cbfifo_enqueue(str, 65, RX_Buffer) == 65);
assert(cbfifo_enqueue(str+65, cap,RX_Buffer) == (cap-65));
assert(cbfifo_length(RX_Buffer) == cap);
assert(cbfifo_dequeue(buf, cap,RX_Buffer) == cap);
assert(cbfifo_length(RX_Buffer) == 0);
assert(strncmp(buf, str, cap) == 0);

// write more than capacity
assert(cbfifo_enqueue(str, 65, TX_Buffer) == 65);
assert(cbfifo_enqueue(str+65, cap,TX_Buffer) == (cap-65));
assert(cbfifo_length(TX_Buffer) == cap);
assert(cbfifo_dequeue(buf, cap,TX_Buffer) == cap);
assert(cbfifo_length(TX_Buffer) == 0);
assert(strncmp(buf, str, cap) == 0);


// write zero bytes
assert(cbfifo_enqueue(str, 0, RX_Buffer) == 0);
assert(cbfifo_length(RX_Buffer) == 0);

// write zero bytes
assert(cbfifo_enqueue(str, 0, TX_Buffer) == 0);
assert(cbfifo_length(TX_Buffer) == 0);

// Exercise the following conditions:
//    enqueue when read < write:
```

```
  //        bytes < CAP-write  (1)
  //        bytes exactly CAP-write  (2)
  //        bytes > CAP-write but < space available (3)
  //        bytes exactly the space available (4)
  //        bytes > space available (5)
  assert(cbfifo_enqueue(str, 32, RX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(RX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, RX_Buffer) == 16);
  assert(cbfifo_length(RX_Buffer) == 16);
  assert(strncmp(buf, str, 16) == 0);


  assert(cbfifo_enqueue(str, 32, TX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(TX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, TX_Buffer) == 16);
  assert(cbfifo_length(TX_Buffer) == 16);
  assert(strncmp(buf, str, 16) == 0);


  assert(cbfifo_enqueue(str+32, 32, RX_Buffer) == 32);  // (1)
  assert(cbfifo_length(RX_Buffer) == 48);
  assert(cbfifo_enqueue(str+64, cap-64,RX_Buffer) == cap-64);  // (2)
  assert(cbfifo_length(RX_Buffer) == cap-16);
  assert(cbfifo_dequeue(buf+16, cap-16, RX_Buffer) == cap-16);
  assert(strncmp(buf, str, cap) == 0);


  assert(cbfifo_enqueue(str, 32, RX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(RX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, RX_Buffer) == 16);
  assert(cbfifo_length(RX_Buffer) == 16);
  assert(strncmp(buf, str, 16) == 0);


  assert(cbfifo_enqueue(str+32, 32, TX_Buffer) == 32);  // (1)
```

```
  assert(cbfifo_length(TX_Buffer) == 48);
  assert(cbfifo_enqueue(str+64, cap-64,TX_Buffer) == cap-64);  // (2)
  assert(cbfifo_length(TX_Buffer) == cap-16);
  assert(cbfifo_dequeue(buf+16, cap-16, TX_Buffer) == cap-16);
  assert(strncmp(buf, str, cap) == 0);

  assert(cbfifo_enqueue(str, 32, TX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(TX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, TX_Buffer) == 16);
  assert(cbfifo_length(TX_Buffer) == 16);
  assert(strncmp(buf, str, 16) == 0);

  assert(cbfifo_enqueue(str+32, cap-20, RX_Buffer) == cap-20);  // (3)
  assert(cbfifo_length(RX_Buffer) == cap-4);
  assert(cbfifo_dequeue(buf, cap-8, RX_Buffer) == cap-8);
  assert(strncmp(buf, str+16, cap-8) == 0);
  assert(cbfifo_length(RX_Buffer) == 4);
  assert(cbfifo_dequeue(buf, 8, RX_Buffer) == 4);
  assert(strncmp(buf, str+16+cap-8, 4) == 0);
  assert(cbfifo_length(RX_Buffer) == 0);

  assert(cbfifo_enqueue(str, 49 ,RX_Buffer) == 49);  // advance so that
read < write
  assert(cbfifo_length(RX_Buffer) == 49);
  assert(cbfifo_dequeue(buf, 16, RX_Buffer) == 16);
  assert(cbfifo_length(RX_Buffer) == 33);
  assert(strncmp(buf, str, 16) == 0);

  assert(cbfifo_enqueue(str+32, cap-20, TX_Buffer) == cap-20);  // (3)
  assert(cbfifo_length(TX_Buffer) == cap-4);
  assert(cbfifo_dequeue(buf, cap-8, TX_Buffer) == cap-8);
  assert(strncmp(buf, str+16, cap-8) == 0);
  assert(cbfifo_length(TX_Buffer) == 4);
```

```
  assert(cbfifo_dequeue(buf, 8, TX_Buffer) == 4);
  assert(strncmp(buf, str+16+cap-8, 4) == 0);
  assert(cbfifo_length(TX_Buffer) == 0);

  assert(cbfifo_enqueue(str, 49 ,TX_Buffer) == 49);  // advance so that
read < write
  assert(cbfifo_length(TX_Buffer) == 49);
  assert(cbfifo_dequeue(buf, 16, TX_Buffer) == 16);
  assert(cbfifo_length(TX_Buffer) == 33);
  assert(strncmp(buf, str, 16) == 0);

  assert(cbfifo_enqueue(str+49, cap-33, RX_Buffer) == cap-33);  // (4)
  assert(cbfifo_length(RX_Buffer) == cap);
  assert(cbfifo_dequeue(buf, cap, RX_Buffer) == cap);
  assert(cbfifo_length(RX_Buffer) == 0);
  assert(strncmp(buf, str+16, cap) == 0);

  assert(cbfifo_enqueue(str, 32, RX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(RX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, RX_Buffer) == 16);
  assert(cbfifo_length(RX_Buffer) == 16);
  assert(strncmp(buf, str, 16) == 0);

  assert(cbfifo_enqueue(str+49, cap-33, TX_Buffer) == cap-33);  // (4)
  assert(cbfifo_length(TX_Buffer) == cap);
  assert(cbfifo_dequeue(buf, cap, TX_Buffer) == cap);
  assert(cbfifo_length(TX_Buffer) == 0);
  assert(strncmp(buf, str+16, cap) == 0);

  assert(cbfifo_enqueue(str, 32, TX_Buffer) == 32);  // advance so that
read < write
  assert(cbfifo_length(TX_Buffer) == 32);
  assert(cbfifo_dequeue(buf, 16, TX_Buffer) == 16);
```

```
assert(cbfifo_length(TX_Buffer) == 16);
assert(strncmp(buf, str, 16) == 0);

assert(cbfifo_enqueue(str+32, cap, RX_Buffer) == cap-16);  // (5)
assert(cbfifo_dequeue(buf, 1, RX_Buffer) == 1);
assert(cbfifo_length(RX_Buffer) == cap-1);
assert(cbfifo_dequeue(buf+1, cap-1,RX_Buffer) == cap-1);
assert(cbfifo_length(RX_Buffer) == 0);
assert(strncmp(buf, str+16, cap) == 0);

assert(cbfifo_enqueue(str+32, cap, TX_Buffer) == cap-16);  // (5)
assert(cbfifo_dequeue(buf, 1, TX_Buffer) == 1);
assert(cbfifo_length(TX_Buffer) == cap-1);
assert(cbfifo_dequeue(buf+1, cap-1,TX_Buffer) == cap-1);
assert(cbfifo_length(TX_Buffer) == 0);
assert(strncmp(buf, str+16, cap) == 0);

//    enqueue when write < read:
//        bytes < read-write (6)
//        bytes exactly read-write (= the space available) (7)
//        bytes > space available (8)

int wpos=0, rpos=0;
assert(cbfifo_enqueue(str, cap-4, RX_Buffer) == cap-4);
wpos += cap-4;
assert(cbfifo_length(RX_Buffer) == cap-4);
assert(cbfifo_dequeue(buf, 32, RX_Buffer) == 32);
rpos += 32;
assert(cbfifo_length(RX_Buffer) == cap-36);
assert(strncmp(buf, str, 32) == 0);
assert(cbfifo_enqueue(str+wpos, 12, RX_Buffer) == 12);
wpos += 12;
assert(cbfifo_length(RX_Buffer) == cap-24);
```

```
assert(cbfifo_enqueue(str, cap-4, TX_Buffer) == cap-4);
wpos += cap-4;
assert(cbfifo_length(TX_Buffer) == cap-4);
assert(cbfifo_dequeue(buf, 32, TX_Buffer) == 32);
rpos += 32;
assert(cbfifo_length(TX_Buffer) == cap-36);
assert(strncmp(buf, str, 32) == 0);
assert(cbfifo_enqueue(str+wpos, 12, TX_Buffer) == 12);
wpos += 12;
assert(cbfifo_length(TX_Buffer) == cap-24);

assert(cbfifo_enqueue(str+wpos, 16, RX_Buffer) == 16);  // (6)
assert(cbfifo_length(RX_Buffer) == cap-8);
assert(cbfifo_dequeue(buf, cap, RX_Buffer) == cap-8);
assert(cbfifo_length(RX_Buffer) == 0);

// reset
wpos=0;
rpos=0;
assert(cbfifo_enqueue(str, cap-4, RX_Buffer) == cap-4);
wpos += cap-4;
assert(cbfifo_length(RX_Buffer) == cap-4);
assert(cbfifo_dequeue(buf, 32, RX_Buffer) == 32);
rpos += 32;
assert(cbfifo_length(RX_Buffer) == cap-36);
assert(strncmp(buf, str, 32) == 0);
assert(cbfifo_enqueue(str+wpos, 12, RX_Buffer) == 12);
wpos += 12;
assert(cbfifo_length(RX_Buffer) == cap-24);

assert(cbfifo_enqueue(str+wpos, 16, TX_Buffer) == 16);  // (6)
```

```
assert(cbfifo_length(TX_Buffer) == cap-8);
assert(cbfifo_dequeue(buf, cap, TX_Buffer) == cap-8);
assert(cbfifo_length(TX_Buffer) == 0);

// reset
wpos=0;
rpos=0;
assert(cbfifo_enqueue(str, cap-4, TX_Buffer) == cap-4);
wpos += cap-4;
assert(cbfifo_length(TX_Buffer) == cap-4);
assert(cbfifo_dequeue(buf, 32, TX_Buffer) == 32);
rpos += 32;
assert(cbfifo_length(TX_Buffer) == cap-36);
assert(strncmp(buf, str, 32) == 0);
assert(cbfifo_enqueue(str+wpos, 12, TX_Buffer) == 12);
wpos += 12;
assert(cbfifo_length(TX_Buffer) == cap-24);


assert(cbfifo_enqueue(str+wpos, 24, RX_Buffer) == 24);  // (7)
assert(cbfifo_length(RX_Buffer) == cap);
assert(cbfifo_dequeue(buf, cap, RX_Buffer) == cap);
assert(cbfifo_length(RX_Buffer) == 0);
assert(strncmp(buf, str+rpos, cap) == 0);

// reset
wpos=0;
rpos=0;
assert(cbfifo_enqueue(str, cap-4, RX_Buffer) == cap-4);
wpos += cap-4;
assert(cbfifo_length(RX_Buffer) == cap-4);
assert(cbfifo_dequeue(buf, 32, RX_Buffer) == 32);
rpos += 32;
```

```
  assert(cbfifo_length(RX_Buffer) == cap-36);
  assert(strncmp(buf, str, 32) == 0);
  assert(cbfifo_enqueue(str+wpos, 12, RX_Buffer) == 12);
  wpos += 12;
  assert(cbfifo_length(RX_Buffer) == cap-24);

  assert(cbfifo_enqueue(str+wpos, 64, RX_Buffer) == 24);  // (8)
  assert(cbfifo_length(RX_Buffer) == cap);
  assert(cbfifo_dequeue(buf, cap, RX_Buffer) == cap);
  assert(cbfifo_length(RX_Buffer) == 0);
  assert(strncmp(buf, str+rpos, cap) == 0);

  assert(cbfifo_enqueue(str+wpos, 64, TX_Buffer) == 24);  // (8)
  assert(cbfifo_length(TX_Buffer) == cap);
  assert(cbfifo_dequeue(buf, cap, TX_Buffer) == cap);
  assert(cbfifo_length(TX_Buffer) == 0);
  assert(strncmp(buf, str+rpos, cap) == 0);
}

/*
 * test_cbfifo.h - tests for cbfifo
 *
 * Author: Howdy Pierce, howdy.pierce@colorado.edu
 *            Kevin Tom, kevn.tom@colorado.edu
 *
 */

#ifndef _TEST_CBFIFO_H_
#define _TEST_CBFIFO_H_

void test_cbfifo();

#endif // _TEST_CBFIFO_H_
```

```
/*
 * init_UART.c
 *
 *  Created on: 04-Nov-2021
 *      Author: Kevin Tom
 *
 * Description
 * -----------
 * This file handles the UART0 configuration, UART0 Interrupt handler and __sys_readc,__sys_write
 *
 *
 */



#include <stdio.h>
#include ”UART.h”
#include ”cbfifo.h”




/* This function initializes UART0 with sysclock.c configuration and enables
 * interrupts on it.
 *
 *  This code is inspired from,
 *   https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter_8
 *
 * Parameters:
 *         uint32_t baud_rate - The baud rate to which it should be configured to
 *
```

```
 * Returns:
 *         None
 * */
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;  //0100 0000 0000 11th bit
      SIM->SCGC5  |=  SIM_SCGC5_PORTA_MASK;    //UART0  is  in  PORTA  ->
PTA1,PTA2 , Alternate Function 1

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

      // Set UART clock to 48 MHz clock
      SIM->SOPT2  |=  SIM_SOPT2_UART0SRC(1);  //MCGFLLCLK  or  MCGPLLCLK/2
(We selected FLL clock in sysclock)


      // Set pins to UART0 Rx and Tx
      //Interrupt status flag (ISF) on  Pin-1 , interrupt is detected,
Alternate 2 (UART)
      PORTA->PCR[1] = PORT_PCR_ISF_MASK | PORT_PCR_MUX(2); // Rx
       //Interrupt status flag (ISF) on  Pin-2 , interrupt is detected,
Alternate 2 (UART)
      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; //Clearing 4 bit BDH
      UART0->BDH |= UART0_BDH_SBR(sbr>>8); //Putting the 9th bit in BDH,
since other 3 bits are 0
```

```
	UART0->BDL = UART0_BDL_SBR(sbr); //Putting rest of 8bits in BDL
	UART0->C4 |= UART0_C4_OSR(UART_OVERSAMPLE_RATE-1); //Over-sampling
15


	// Disable interrupts for RX active edge and LIN break detect,
select one stop bit
	UART0->BDH |= UART0_BDH_RXEDGIE(0) | UART0_BDH_SBNS(STOP_BITS) |
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(DATA_SIZE) |
UART0_C1_PE(PARITY);
	// 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);


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



	/******************Interrupts                    Configuration
part*********************/
	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;

}
```

```
/* Interrupt handler for UART0, This function will enqueue and dequeue
from
 * cbfifo on interrupt triggers.
 *
 *  This code is inspired from,
 *    https://github.com/alexander-g-
dean/ESF/blob/master/NXP/Code/Chapter_8
 *
 * Parameters:
 *         None
 *
 * Returns:
 *         None
 * */
void UART0_IRQHandler(void)
{
	uint8_t ch;


	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) //RDRF is '1': Receive data
buffer is full
	{
		ch = UART0->D; //Reading the received character


		UART0->D = ch; //Echoing back the character


		if(cbfifo_enqueue(&ch,1,RX_Buffer) == 1)
		{
			;
		}
		else
		{
			//If not discard
		}
	}


	if ( (UART0->C2 & UART0_C2_TIE_MASK) &&    // transmitter interrupt
enabled
			(UART0->S1 & UART0_S1_TDRE_MASK) ) //If transmit data
buffer is empty
	{
		// checking if tx buffer empty
		if(cbfifo_length(TX_Buffer) > 0)
		{
```

```
                cbfifo_dequeue(&ch, 1, TX_Buffer);
                UART0->D = ch; //Put the data across UART line
            }
            else
            {
                // queue is empty so disable transmitter interrupt
                UART0->C2 &= ~UART0_C2_TIE_MASK;
            }
        }
}
```

```
/* overwriting the sys_write function which will help in using printf,
when we
 * overwrite this function we will direct printf to UART0.
 *
 *  This code is inspired from,
 *   https://github.com/alexander-g-
dean/ESF/blob/master/NXP/Code/Chapter_8
 *
 * Parameters:
 *                    int handle -  stdout - (handle = 1)
 *                                          stderr - (handle = 2)
 *                                          In this case we are directing
both to serial, so no need to
 *                                          take care of this
 *                char *buf  -  start address of buffer to be written
 *                int size   -  number of characters to be written
 *
 * Returns:
 *         -1 in case of error
 *          0 in case of sucess
 * */
```

```
int __sys_write(int handle, char *buf, int size)
{
	//In case of error return -1
	if(buf == NULL || size <= 0)
		return -1;

	//if(size > (256 - cbfifo_length(TX_Buffer)))
	while( size > (256 - cbfifo_length(TX_Buffer)) );


	//Enqueue to transmit buffer
	if(cbfifo_enqueue(buf,size,TX_Buffer) != size)
		return -1;

	//Generating a transmit signal
	//start transmitter if it isn't already running
	if (!(UART0->C2 & UART0_C2_TIE_MASK))
	{
		UART0->C2 |= UART0_C2_TIE(1);
	}


	//If success return 0
	return 0;
}
```

```
/* overwriting the__sys_readc function which will help in using getchar(), when we
 * overwrite this function we will direct pgetchar to UART0.
 *  This code is inspired from,
 *   https://github.com/alexander-g-dean/ESF/blob/master/NXP/Code/Chapter_8
 *
 * Parameters:
 *              None
 *
 * Returns:
 *         int - ASCII code of character written/0 in case of error
 * */
int __sys_readc(void)
{
     char c;
     //Wait till something is written into RX_Buffer, i.e, wait till use writes something
     while((cbfifo_length(RX_Buffer)==0));

     //After RX_Buffer has value dequeue it and return
     if(cbfifo_dequeue(&c, 1, RX_Buffer) == 1)
          return c;
     else
          return 0;
}

#ifndef UART_H
#define UART_H

#include <stdint.h>
#include <MKL25Z4.H>
```

```
#define UART_OVERSAMPLE_RATE     (16)
#define BUS_CLOCK                     (24e6)
#define SYS_CLOCK                     (24e6)



//Parameters according to assignment
#define STOP_BITS               (2) // 1 : One Stop bit       | 2 : Two
stop bits
#define DATA_SIZE                     (0) // 0 : 8-bit mode         | 1
: for 9-bit mode
#define PARITY                        (0) // 0 : Parity is disabled | 1
: Parity is enabled



void Init_UART0(uint32_t baud_rate);


#endif
```