ECEN 5813

PROJECT 2

TEAM MEMBERS:
1) Preshit Harlikar 2)Shivam Khandelwal

CIRCULAR BUFFER

1.circbuf.c

```
/*****************************************************************************
**************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file circbuf.c
 * @brief This file includes Circular Buffer functions
 * @date October 20, 2017
 *
 * long description - The circbuf.c file includes functions to -
 *                      1) add data to circular
buffer(CB_buffer_add_item())
 *                      2) remove data from circular buffer
(CB_buffer_remove_item())
 *                      3) check whether or not buffer is
full(CB_is_full())
 *                      4) check whether or not buffer is
empty(CB_is_empty())
 *                      5) peek at a location from head (CB_peek())
 *                      6) initialize circular buffer (CB_init())
 *                      7) destroy a circular buffer (CB_destroy())
 *
 *

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


/*----- Header-Files --------------------------------------------------
---*/

#include "circbuf.h"


/********************************** CB_buffer_add_item()
*************************************************
 *
 * @name   -  CB_buffer_add_item()
 * @brief  -  function to add data to circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *data : data to be added to buffer
 *
```

```
 * long description - This function adds data to circular buffer from a
given memory location.
 *
 * @return -  BUFFER_FULL : if buffer is full
 * @return -  SUCCESS : if data added successfully to buffer
 *

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

/******************************** CB_buffer_add_item() function
definition *********************************/

CB_status CB_buffer_add_item(CB_t *cb, uint8_t *data)
{
    if(cb->buffer==NULL)
    {
        return NULL_ERROR;
    }

    else
    {
        if(cb->count==cb->length)
        {
            return BUFFER_FULL;
        }
            else if(cb->head==cb->buffer_end && cb->count<cb->length)
        {
            cb->head=cb->buffer;
            *(cb->head)=*data;
            cb->count++;
            cb->head++;
            return SUCCESS;
        }
        else
        {
            *(cb->head)=*data;
            cb->count++;
            cb->head++;
            return SUCCESS;
        }
    }
}

/******************************** CB_buffer_remove_item()
*************************************************
 *
 * @name   -  CB_buffer_remove_item()
 * @brief  -  function to remove data from circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *data : location to which data is dumped from buffer.
 *
 * long description - This function removes data from circular buffer to
a given memory location.
```

```
 *
 * @return -  BUFFER_EMPTY : if buffer is empty
 * @return -  SUCCESS : if data removed successfully from buffer
 *
*****************************************************************************
****************************/

/******************************** CB_buffer_remove_item() function
definition *********************************/


CB_status CB_buffer_remove_item(CB_t *cb, uint8_t *data)
{
    if(cb->buffer==NULL)
    {
        return NULL_ERROR;
    }
    else
    {
         if(cb->count==0)
        {
            return BUFFER_EMPTY;
        }

        else if(cb->tail==cb->buffer_end)
        {
            *data=*(cb->tail);
            cb->tail=cb->buffer;
            cb->count--;
            return SUCCESS;
        }
        else
        {
            *data=*(cb->tail);
            cb->tail++;
            cb->count--;
            return SUCCESS;
        }
    }
}

/********************************** CB_is_full()
*************************************************
 *
 * @name   -  CB_is_full()
 * @brief  -  function to check whether or not buffer is full
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function checks whether or not buffer is full.
 *
 * @return -  BUFFER_FULL : if buffer is full.
 * @return -  NULL_ERROR : if buffer is not full.
 *
```

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

/******************************** CB_is_full() function definition
*********************************/


CB_status CB_is_full(CB_t *cb)
{
    if(cb->buffer==NULL)
    {
        return NULL_ERROR;
    }
    else
    {
        if (cb->count == cb->length)
            return BUFFER_FULL;
        else
            return BUFFER_NOT_FULL;
    }

}

/********************************* CB_is_empty()
*************************************************
 *
 * @name   -  CB_is_empty()
 * @brief  -  function to check whether or not buffer is empty
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function checks whether or not buffer is
empty.
 *
 * @return -  BUFFER_EMPTY : if buffer is empty.
 * @return -  NULL_ERROR : if buffer is not empty.
 *

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

/******************************** CB_is_empty() function definition
*********************************/


CB_status CB_is_empty(CB_t *cb)
{
    if(cb->buffer==NULL)
    {
        return NULL_ERROR;
    }
    else
    {
        if (cb->count == 0)
```

```
            return BUFFER_EMPTY;
        else
            return BUFFER_NOT_EMPTY;
    }

}

/********************************** CB_peek()
************************************************
 *
 * @name   -  CB_peek()
 * @brief  -  peek at a location from head
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *peek_ptr : location to which data is dumped from buffer.
 * @param  -  peek_pos : position to peek from head
 *
 * long description - This function peeks at a location from head. The
data of the peeked location
 *                             is copied to peek pointer.
 *
 * @return -  BUFFER_EMPTY : if buffer is empty
 * @return -  SUCCESS : if data removed successfully from buffer
 * @return -  PEEK_LENGTH_ERROR : if data removed successfully from
buffer
 *

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

/********************************** CB_peek() function definition
***********************************/


CB_status CB_peek(CB_t *cb, uint8_t peek_pos, uint8_t *peek_ptr)
{
    if(((cb->count) != 0) && (peek_pos <= (cb->count)))
    {
        if(((cb->head+peek_pos))<(cb->buffer_end))
        {
            *peek_ptr= *cb->head+peek_pos;
        }

        else if(((cb->head)+(peek_pos))>=(cb->buffer_end))
        {
            *peek_ptr= *((cb->buffer)+((cb->buffer_end)-((cb-
>head+peek_pos))));
        }

      return SUCCESS;
    }
    else if((cb->count) == 0)
    {
        return BUFFER_EMPTY;
    }
```

```
    else
    {
        return PEEK_LENGTH_ERROR;
    }
}

/********************************** CB_init()
*************************************************
 *
 * @name   -  CB_init()
 * @brief  -  function to create a circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  length : size of buffer in bytes.
 *
 * long description - This function creates a circular buffer of
specified bytes.
 *
 * @return -  SUCCESS : if buffer is successfully created.
 *

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

/******************************** CB_init() function definition
***********************************/

CB_status CB_init(CB_t *cb, uint8_t length)
{
    if(cb==NULL)
    {
      return NULL_ERROR;
    }
    else
    {
      cb->buffer = (uint8_t*)malloc(sizeof(uint8_t)*length);
      if(cb->buffer==NULL)
      {
              return NULL_ERROR;
      }
      else
      {
              cb->buffer_end = cb->buffer + (sizeof(uint8_t)*length);
              cb->head = cb->buffer;
              cb->tail = cb->buffer;
              cb->count = 0;
              cb->length = length;
              return SUCCESS;
      }
    }
}

/********************************** CB_destroy()
*************************************************
 *
```

```
 * @name   -  CB_destroy()
 * @brief  -  function to destroy a circular buffer
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function destroys a circular buffer.
 *
 * @return -  SUCCESS : if buffer is successfully destroyed.
 *
*****************************************************************************
***************************/

/******************************** CB_destroy() function definition
*********************************/

CB_status CB_destroy(CB_t *cb)
{
    if(cb->buffer==NULL)
    {
        return NULL_ERROR;
    }
    else
    {
        free(cb->buffer);
        return SUCCESS;
    }
}
```

2.circbuf.h

```
/*****************************************************************************
*************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file circbuf.h
 * @brief This  header file includes Circular Buffer functions
declarations
 * @date October 20, 2017
 *
 * long description - The circbuf.c file includes functions declarations
to -
 *                      1) add data to circular
buffer(CB_buffer_add_item())
 *                      2) remove data from circular buffer
(CB_buffer_remove_item())
 *                      3) check whether or not buffer is
full(CB_is_full())
 *                      4) check whether or not buffer is
empty(CB_is_empty())
 *                      5) peek at a location from head (CB_peek())
 *                      6) initialize circular buffer (CB_init())
 *                      7) destroy a circular buffer (CB_destroy())
```

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

#ifndef INCLUDES_CIRCBUF_H_
#define INCLUDES_CIRCBUF_H_

/*----- Header-Files -------------------------------------------------------
---*/
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>

/*----- Circular Buffer Struct ---------------------------------------------
--*/

typedef struct{
uint8_t *buffer;
uint8_t *buffer_end;
uint8_t *head;
uint8_t *tail;
uint8_t length;
uint8_t count;
}CB_t;

/*----- Enum for Circular Buffer and UART ----------------------------------
--*/
typedef enum{
BUFFER_FULL=1,
BUFFER_NOT_FULL=2,
BUFFER_EMPTY=3,
BUFFER_NOT_EMPTY=4,
SUCCESS=5,
FAILURE=6,
NULL_ERROR=7,
PEEK_LENGTH_ERROR=8,
TX_SUCCESS=9,
RX_SUCCESS=10,
TX_IRQ=11,
RX_IRQ=12
}CB_status;

/********************************** CB_buffer_add_item()
*************************************************
 *
 * @name   -  CB_buffer_add_item()
 * @brief  -  function to add data to circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *data : data to be added to buffer
 *
 * long description - This function adds data to circular buffer from a
given memory location.
```

```
 *
 * @return -  BUFFER_FULL : if buffer is full
 * @return -  SUCCESS : if data added successfully to buffer
 *

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

/******************************** CB_buffer_add_item() function
declaration *********************************/

CB_status CB_buffer_add_item(CB_t *cb, uint8_t *data);


/********************************** CB_buffer_remove_item()
***************************************************
 *
 * @name   -  CB_buffer_remove_item()
 * @brief  -  function to remove data from circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *data : location to which data is dumped from buffer.
 *
 * long description - This function removes data from circular buffer to
a given memory location.
 *
 * @return -  BUFFER_EMPTY : if buffer is empty
 * @return -  SUCCESS : if data removed successfully from buffer
 *

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

/******************************** CB_buffer_remove_item() function
declaration *********************************/

CB_status CB_buffer_remove_item(CB_t *cb, uint8_t *data);

/********************************** CB_is_full()
***************************************************
 *
 * @name   -  CB_is_full()
 * @brief  -  function to check whether or not buffer is full
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function checks whether or not buffer is full.
 *
 * @return -  BUFFER_FULL : if buffer is full.
 * @return -  NULL_ERROR : if buffer is not full.
 *

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

```
/********************************* CB_is_full() function declaration
**********************************/

CB_status CB_is_full(CB_t *cb);

/********************************** CB_is_empty()
***************************************************
 *
 * @name   -  CB_is_empty()
 * @brief  -  function to check whether or not buffer is empty
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function checks whether or not buffer is
empty.
 *
 * @return -  BUFFER_EMPTY : if buffer is empty.
 * @return -  NULL_ERROR : if buffer is not empty.
 *

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

/********************************* CB_is_empty() function declaration
**********************************/

CB_status CB_is_empty(CB_t *cb);

/********************************** CB_peek()
***************************************************
 *
 * @name   -  CB_peek()
 * @brief  -  peek at a location from head
 * @param  -  *cb : pointer to circular buffer
 * @param  -  *peek_ptr : location to which data is dumped from buffer.
 * @param  -  peek_pos : position to peek from head
 *
 * long description - This function peeks at a location from head. The
data of the peeked location
 *                             is copied to peek pointer.
 *
 * @return -  BUFFER_EMPTY : if buffer is empty
 * @return -  SUCCESS : if data removed successfully from buffer
 * @return -  PEEK_LENGTH_ERROR : if data removed successfully from
buffer
 *

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

/********************************* CB_peek() function declaration
**********************************/


CB_status CB_peek(CB_t *cb, uint8_t peek_pos, uint8_t *peek_ptr);
```

```
/********************************** CB_init()
************************************************
 *
 * @name   -  CB_init()
 * @brief  -  function to create a circular buffer
 * @param  -  *cb : pointer to circular buffer
 * @param  -  length : size of buffer in bytes.
 *
 * long description - This function creates a circular buffer of
specified bytes.
 *
 * @return -  SUCCESS : if buffer is successfully created.
 *

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

/******************************** CB_init() function declaration
***********************************/

CB_status CB_init(CB_t *cb, uint8_t length);

/********************************** CB_destroy()
************************************************
 *
 * @name   -  CB_destroy()
 * @brief  -  function to destroy a circular buffer
 * @param  -  *cb : pointer to circular buffer
 *
 * long description - This function destroys a circular buffer.
 *
 * @return -  SUCCESS : if buffer is successfully destroyed.
 *

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

/******************************** CB_destroy() function declaration
***********************************/

CB_status CB_destroy(CB_t *cb);

#endif /* INCLUDES_CIRCBUF_H_ */
```

3.memory.c

```
/*****************************************************************************
**************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @file memory.c
 * @brief This file includes memory manipulation functions
```

```
 * @date October 1, 2017
 *
 * long decription - The memory.c file includes memory manipulation
functions for -
 *                       1) moving bytes of data from source to
destination(my_memmove())
 *                       2) copying bytes of data from source to
destination(my_memcpy())
 *                       3) setting bytes of data to a specified
value(my_memset())
 *                       4) setting bytes of data to zero(my_memzero())
 *                       5) reversing bytes of data at a specified memory
loaction(my_reverse())
 *                       6) reserving a specified number of words at a
particular
 *                          memory location.(reserve_words())
 *                       7) free a dynamic memory allocation by providing
the pointer src to
 *                          the function(free_words())
 *

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

/************* including standard libraries*************/


#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>


/********************************** my_memmove()
*****************************************
 *
 * @name  *my_memmove(uint8_t * src, uint8_t * dst, size_t length)
 * @brief function to move length of bytes data from source to
destination.
 * @param 1) *src - pointer to a source memory location.
 *        2) *dst - pointer to a destination memory location
 *        3) length - length of data bytes to be moved.
 *
 * long description - This function takes two byte pointers (one source
and one destination)
 *                    and a length of bytes to copy from the source
location (src) to the
 *                    destination(dst). This is done by alloocating a
length of bytes using
 *                    'malloc' function. Data is first copied to an
intermediate location (temp)
 *                    from source location(src). Then the data is copied
from temp to destination
```

```
 *                     location(dst).Thus, the data is copied from source
to destination even if there
 *                     is an overlap. Copy occurs with no data corruption.
Finally, the function
 *                     returns destination location
 *

 *
 * @return dst - destination location.
 *

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

 /***************************** my_memmove() function definition
*********************************/


uint8_t *my_memmove(uint8_t * src, uint8_t * dst, size_t length)
{
    if (src==NULL)
        return NULL;
    else
    {
    size_t i; // variable i declared for loop condition
    uint8_t *temp=malloc(length); // length of bytes allocated starting
from temp
    for(i=0;i<length;i++) // loop condition to copy bytes of data from
src to temp
    {
        *(temp+i)=*(src+i);//Copying each byte consecutively.
    }

    for(i=0;i<length;i++) // loop condition to copy bytes of data from
temp to dst.
    {
        *(dst+i)=*(temp+i); // Copying each byte consecutively.
    }
    return dst; // Returning destination location.
    }
}


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


/********************************** my_memcpy()
************************************************
 *
 * @name  *my_memmove(uint8_t * src, uint8_t * dst, size_t length)
 * @brief function to move length of bytes data from source to
destination.
 * @param 1) *src - pointer to a source memory location.
```

```
 *        2) *dst - pointer to a destination memory location
 *        3) length - length of data bytes to be moved.
 *
 * long description - This function takes two byte pointers (one source
and one destination)
 *                    and a length of bytes to copy from the source
location (src) to the
 *                    destination(dst). Data is copied from source
location(src) to destination
 *                    location(dst).Is there is an overlap in source(src)
and detination(dst)
 *                    memory locations, copy occurs but the data
corrupts. Finally, the function
 *                    returns destination location
 *

 *
 * @return dst - destination location.
 *

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

/*************************** my_memcpy() function definition
***********************************/

uint8_t * my_memcpy(uint8_t * src, uint8_t * dst, size_t length)
{
    int i; // variable i declared for loop condition.
    if (src == dst) // check if source and destination are same.
    {
        return dst;//return destination location.
    }
    else if(src > dst) // check if source location is before destination
location.
    {
        for( i = 0; i < length-1; i ++) // loop condition to copy data
from src to dst
        {
            *(dst+i) = *(src+i); // copying each byte of data
consecutively from src to dst.

        }

    }
    else if(src < dst) // check if source location is after destination
location.
    {
        for ( i = length-1; i >= 0; i--) // loop condition to copy data
from src to dst
        {
            *(dst+i) = *(src+i); // copying each byte of data
consecutively from src to dst.
        }
```

```
    }
    return dst; //return destination location.
}

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

/********************************* my_memset()
*********************************************
 *
 * @name  my_memset(uint8_t * src, size_t length, uint8_t value)
 * @brief function to set a length of  data bytes to a specified value
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be set to the specified
value.
 *        3) value - value to be set for each byte
 *
 * long description - This function takes a pointer to a source memory
location(src) and
 *                    a consecutive length in bytes are set to the
specified value. The function
 *                    returns a pointer to the source(src).
 *
 * @return src - source location.
 *

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

/*************************** my_memset() function definition
***********************************/

uint8_t * my_memset(uint8_t * src, size_t length, uint8_t value)
{
    int i; // variable i declared for loop condition
    if(src==NULL)
       return NULL;
    else
    {
      for(i=0;i<length;i++) // loop condition to set a length of bytes to
specified value.
      {
            *(src+i)=value; //  setting each byte consecutively to the
specified value.
      }
      return src;
    }
}


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

```
/********************************** my_memzero()
*********************************************
 *
 * @name  my_memzero(uint8_t * src, size_t length)
 * @brief function to set a length of  data bytes zero.
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be set to zero.
 *
 * long description - This function takes a pointer to a source memory
location(src) and
 *                    a consecutive length in bytes are set to zero. The
function
 *                    returns a pointer to the source(src).
 *
 * @return src - source location.
 *

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

/***************************** my_memzero() function definition
**********************************/


uint8_t * my_memzero(uint8_t * src, size_t length)
{
    int i; // variable i declared for loop condition
    if (src==NULL)
      return NULL;

    else
    {
    for(i=0;i<length;i++) // loop condition to set a length of bytes to
zero.
    {
        *(src+i)=0; //  setting each byte consecutively to zero.
    }
    return src;
    }
}

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

/********************************** my_reverse()
*********************************************
 *
 * @name  my_reverse(uint8_t * src, size_t length)
 * @brief function to reverse a length of data bytes.
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be reversed.
 *
 * long description - This function takes a pointer to a source memory
location(src) and
```

```
 *                    a consecutive length in bytes are reversed. The
function
 *                    returns a pointer to the source(src) if
successfully executed.
 *
 * @return src - source location if reverse is successful else returns 0.
 *

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

/*************************** my_reverse() function definition
**********************************/


uint8_t * my_reverse(uint8_t * src, size_t length)
{

    if(src == NULL)
      return NULL;

    else
    {
    int i; // variable i declared for loop condition.
    size_t j=length; // variable j declared and initialized to length
    int k=0; // variable k declared and initialized to zero for checking
length
    uint8_t *temp=malloc(j*sizeof(size_t)); // memory of size of integer
value of j allocated.

    for(i=0;i<length;i++)
    {
        *(temp+i)=*(src+i); //consecutively copying each data byte from
src to temp
    }

    for(i=0;i<length;i++)
    {
        j--;
        *(src+j)=*(temp+i); //consecutively copying each data byte from
temp to src.
    }

    j=length; // assigning j the value of length

    for(i=0;i<length;i++)
    {
        j--;
        if(*(src+i)==*(temp+j)) //comparing data in reversed bytes
        {
            k++; // incrementing k for each successful comparison.
        }
    }
```

```
    if (k==length)
    {
        return src; // returning source location if data successfully
reversed.
    }
    else return 0; // returning zero if not reversed successfully.
    }
}

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

/********************************** reserve_words()
***********************************************
 *
 * @name  reserve_words(size_t length)
 * @brief function to reserve a length of bytes.
 * @param length - length of bytes to be reserved.
 *
 *
 * long description - This function takes a pointer to a memory
location(temp) and
 *                    a specified length in bytes are reserved using
'malloc' function.
 *                    The function returns a pointer to the memory
location(temp).
 *
 * @return temp - memory location for reserved bytes.
 *

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

/***************************** reserve_words() function definition
***********************************/


uint8_t * reserve_words(size_t length)
{
    uint8_t *temp = malloc(length*sizeof(size_t)); // memory of length
number of bytes allocated
                                                   // starting from temp
location.

    return (temp); // returning temp memory location.
}


/********************************** free_words()
***********************************************
 *
 * @name  free_words(uint32_t * src)
 * @brief function to free a dynamic memory allocation.
 * @param *src - pointer to source memory location.
```

```
 *
 *
 * long description - This function frees a dynamic memory allocation by
providing the pointer src to
 *                    the function
 *
 * @return void
 *

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

/***************************** freee_words() function definition
***********************************/

void free_words(uint32_t * src)
{
    free(src);
}

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

4.memory.h

```
/************************************************************************
**************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @file memory.c
 * @brief This file includes memory manipulation function declarations.
 * @date October 1, 2017
 *
 * long decription - The memory.h file includes memory manipulation
function declarations for -
 *                      1) moving bytes of data from source to
destination(my_memmove())
 *                      2) copying bytes of data from source to
destination(my_memcpy())
 *                      3) setting bytes of data to a specified
value(my_memset())
 *                      4) setting bytes of data to zero(my_memzero())
 *                      5) reversing bytes of data at a specified memory
loaction(my_reverse())
 *                      6) reserving a specified number of words at a
particular
 *                         memory location.(reserve_words())
 *                      7) free a dynamic memory allocation by providing
the pointer src to
 *                         the function(free_words())
 *
```

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

#ifndef MEMORY_H_INCLUDED
#define MEMORY_H_INCLUDED

/************* including standard libraries*************/

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <math.h>

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

/********************************* my_memmove()
*****************************************
 *
 * @name  *my_memmove(uint8_t * src, uint8_t * dst, size_t length)
 * @brief function to move length of bytes data from source to
destination.
 * @param 1) *src - pointer to a source memory location.
 *        2) *dst - pointer to a destination memory location
 *        3) length - length of data bytes to be moved.
 *
 * long description - This function takes two byte pointers (one source
and one destination)
 *                    and a length of bytes to copy from the source
location (src) to the
 *                    destination(dst). This is done by alloocating a
length of bytes using
 *                    'malloc' function. Data is first copied to an
intermediate location (temp)
 *                    from source location(src). Then the data is copied
from temp to destination
 *                    location(dst).Thus, the data is copied from source
to destination even if there
 *                    is an overlap. Copy occurs with no data corruption.
Finally, the function
 *                    returns destination location
 *


 *
 * @return dst - destination location.
 *

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


/***************************** my_memmove() function declaration
*********************************/
```

```
uint8_t *my_memmove(uint8_t * src, uint8_t * dst, size_t length);


/********************************** my_memcpy()
************************************************
 *
 * @name  *my_memmove(uint8_t * src, uint8_t * dst, size_t length)
 * @brief function to move length of bytes data from source to
destination.
 * @param 1) *src - pointer to a source memory location.
 *        2) *dst - pointer to a destination memory location
 *        3) length - length of data bytes to be moved.
 *
 * long description - This function takes two byte pointers (one source
and one destination)
 *                    and a length of bytes to copy from the source
location (src) to the
 *                    destination(dst). Data is copied from source
location(src) to destination
 *                    location(dst).Is there is an overlap in source(src)
and detination(dst)
 *                    memory locations, copy occurs but the data
corrupts. Finally, the function
 *                    returns destination location
 *


 *
 * @return dst - destination location.
 *

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

uint8_t * my_memcpy(uint8_t * src, uint8_t * dst, size_t length);


/********************************** my_memset()
************************************************
 *
 * @name  my_memset(uint8_t * src, size_t length, uint8_t value)
 * @brief function to set a length of  data bytes to a specified value
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be set to the specified
value.
 *        3) value - value to be set for each byte
 *
 * long description - This function takes a pointer to a source memory
location(src) and
 *                    a consecutive length in bytes are set to the
specified value. The function
 *                    returns a pointer to the source(src).
 *
 * @return src - source location.
 *
```

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

/*************************** my_memset() function declaration
**********************************/


uint8_t * my_memset(uint8_t * src, size_t length, uint8_t value);

/********************************** my_memzero()
***********************************************
 *
 * @name  my_memzero(uint8_t * src, size_t length)
 * @brief function to set a length of  data bytes zero.
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be set to zero.
 *
 * long description - This function takes a pointer to a source memory
location(src) and
 *                    a consecutive length in bytes are set to zero. The
function
 *                    returns a pointer to the source(src).
 *
 * @return src - source location.
 *

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


/*************************** my_memzero() function declaration
**********************************/

uint8_t * my_memzero(uint8_t * src, size_t length);


/********************************** my_reverse()
***********************************************
 *
 * @name  my_reverse(uint8_t * src, size_t length)
 * @brief function to reverse a length of data bytes.
 * @param 1) *src - pointer to a source memory location.
 *        2) length - length of data bytes to be reversed.
 *
 * long description - This function takes a pointer to a source memory
location(src) and
 *                    a consecutive length in bytes are reversed. The
function
 *                    returns a pointer to the source(src) if
successfully executed.
 *
 * @return src - source location if reverse is successful else returns 0.
 *
```

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


/*************************** my_reverse() function declaration
**********************************/

uint8_t * my_reverse(uint8_t * src, size_t length);


********************************** reserve_words()
************************************************
 *
 * @name  reserve_words(size_t length)
 * @brief function to reserve a length of bytes.
 * @param length - length of bytes to be reserved.
 *
 *
 * long description - This function takes a pointer to a memory
location(temp) and
 *                    a specified length in bytes are reserved using
'malloc' function.
 *                    The function returns a pointer to the memory
location(temp).
 *
 * @return temp - memory location for reserved bytes.
 *

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

/*************************** reserve_words() function declaration
**********************************/

uint8_t * reserve_words(size_t length);

/********************************** free_words()
************************************************
 *
 * @name  free_words(uint32_t * src)
 * @brief function to free a dynamic memory allocation.
 * @param *src - pointer to source memory location.
 *
 *
 * long description - This function frees a dynamic memory allocation by
providing the pointer src to
 *                    the function
 *
 * @return void
 *

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

```
/*************************** freee_words() function declaration
**********************************/

void free_words(uint32_t * src);



#endif // MEMORY_H_INCLUDED
```

5.conversion.c

```
/*************************************************************************
*************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file conversion.c
 * @brief This file includes data conversion functions
 * @date October 2, 2017
 *
 * long decription - The conversion.c file includes data conversion
functions for -
 *                       1) integer to ascii string (my_itoa())
 *                       2) ascii string to 32-bit signed integer
(my_atoi())
 *                       3) little endian to big endian
(little_to_big32())
 *                       4) big endian to little endian
(big_to_little32())
 *
*************************************************************************
************************/

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <math.h>


/********************************** little_to_big32()
*****************************************
 *
 * @name  *little_to_big32(uint32_t *data, uint32_t length)
 * @brief function to convert little endian to big endian.
 * @param 1) *data - pointer to a memory location.
 *        2) length - length of ascii string(including sign).
 *
 * long description - This function converts data stored at memory
locations (from (data) to
 *                    (data + length -1)) from little endian to big
endian format for a 32-bit
```

```
 *                    word size. Each byte value is stored in uint8_t
variable (a1,a2,a3,a4 to
 *                    break and shift the 32-bit data at memory
location((data + b)). After each
 *                    conversion memory location is incremented till the
data of all bytes is
 *                    converted.
 *
 * @return 0 if function is executed successfully.
 *

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

/***************************** little_to_big32 function definition
*******************************/

uint32_t *little_to_big32(uint32_t *data, uint32_t length)
{
    if(data==NULL)
      return NULL;

    else
    {
    uint32_t b = 0; // variable b initialized for loop condition
    while(b < length) //loop condition for data conversion till all 32-
bit words are converted.
    {
        uint32_t a = *(data + b); // variable a initialized to store data
of *(data + b)

        int8_t a1,a2,a3,a4; //variables declared to break and shift 32-
bit data in a
        a1 = (a & 0x000000ff); // a1 stores last byte of a
        a2 = (a & 0x0000ff00) >> 8; // a2 stores 3rd byte of a
        a3 = (a & 0x00ff0000) >> 16; //a3 stores 2nd byte of a
        a4 = (a & 0xff000000) >> 24; // a4 stores 1st byte of a

        a = ((a1<<24)|(a2<<16)|(a3<<8)|(a4)); //data shifted and bitwise
OR operation performed.

        *(data + b) = a; //data after conversion stored in *(data+b)
        b++; //Incrementing b
    }
    return data;
    }
}


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

```
/********************************** big_to_little32()
*****************************************
 *
 * @name  big_to_little32(uint32_t *data, uint32_t length)
 * @brief function to convert big endian to little endian.
 * @param 1) *data - pointer to a memory location.
 *        2) length - length of ascii string(including sign).
 *
 * long description - This function converts data stored at memory
locations (from (data) to
 *                    (data + length -1)) from big endian to little
endian format for a 32-bit
 *                    word size. Each byte value is stored in uint8_t
variable (a1,a2,a3,a4 to
 *                    break and shift the 32-bit data at memory
location((data + b)). After each
 *                    conversion memory location is incremented till the
data of all bytes is
 *                    converted.
 *
 * @return 0 if function is executed successfully.
 *

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

/***************************** big_to_little32 function definition
*******************************/


uint32_t *big_to_little32(uint32_t *data, uint32_t length)
{
    if(data==NULL)
      return NULL;

    else
    {
    uint32_t a = 0;  // variable a initialized for loop condition
    while(a < length)  //loop condition for data conversion till all 32-
bit words are converted.
    {
        uint32_t b = *(data + a); // variable b initialized to store data
of *(data + a)

        int8_t b1,b2,b3,b4;  //variables declared to break and shift 32-
bit data in b
        b1 = (b & 0xff000000) >> 24; // b1 stores 1st byte of b
        b2 = (b & 0x00ff0000) >> 16; // b2 stores 2nd byte of b
        b3 = (b & 0x0000ff00) >> 8; // b3 stores 3rd byte of b
        b4 = (b & 0x000000ff); // b4 stores last byte of b

        b = ((b1)|(b2<<8)|(b3<<16)|(b4<<24)); //data shifted and bitwise
OR operation performed.
```

```
        *(data + a) = b; //data after conversion stored in *(data+a)
        a++; // Incrementing a
    }
    return data;
    }
}


/********************************** my_itoa()
*********************************************************
 *
 * @name  my_itoa(int32_t data,uint8_t* ptr, uint32_t base)
 * @brief function to convert a signed 32-bit integer to an ascii string
and store it at memory loaction
 * @param 1) data - signed 32-bit integer in decimal.
 *        2) *ptr - pointer to a memory location.
 *        3) base - base to which data is converted.
 *
 * long description - This function converts a standard decimal integer
(base 10) to number
 *                    of specified base. The sign of the decimal
number(data)is first determined
 *                    specified memory location (*ptr). The remainder
after taking modulus with base
 *                    and then stored at (using % operator) is converted
to ascii character and
 *                    stored at next memory location (*(ptr + 1)). A
variable 'l' is initialized to
 *                    zero and incremented to determine the length of
string. The decimal number is
 *                    then divided until it becomes zero. After each
modulus operation, ptr is
 *                    incremented and remainder is stored in *ptr. The
characters stored in memory
 *                    locations - (ptr + 1) to (ptr + l) are reversed in
order to store the ascii
 *                    string in correct format. Lastly, the length of the
ascii string (including
 *                    sign) 'l' is returned by the function.
 *
 * @return length of ascii string (uint8_t l)
 *

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

/********************************** my_itoa function definition
*************************************/

void my_itoa(int32_t data,uint8_t* ptr, uint32_t base)
{
    if((base<17)&&(base>1)) // condtion to run for base 2 to 16 only.
    {
```

```
       uint8_t s = 0; // variable s initialized for storing characters at
ptr.
        uint32_t l=0; //  variable l initialized to determine length of
string.

       if(data==0) // condition to check for zero.
       {
           *ptr =48; // ascii value in decimal for '0'
           l=l+1; //incrementing l
           //return l;
       }

       if(data>0) // condition to check for positive number
       {
           *ptr = 43; // ascii value in decimal for '+'
           data = data; // no inversion if data is positive.
           l=l+1; // incrementing l

       }

       else if(data<0) // condition to check for negative number.
       {
           *ptr = 45; // ascii value in decimal for '-'
           data = -data; // inversion if data is negative.
           l=l+1; // incrementing l
       }


       while(data != 0) // condition to check for zero after each
division.
       {
            ptr++; // incrementing ptr to store remainder in next memory
location.
            s = data%base; //taking modulus to obtain new remainder.
            if(s>9) // condition to check if remainder is greater than 9
            {
               s = s - 9; // conversion of remainder to corresponding
ascii value in decimal.
               s = s + 64; // 64 is ascii value of 'A'
            }
            else // condition for remainder between 0-9.
            {
               s = s + 48; // conversion of remainder to corresponding
ascii value in decimal.
            }

            *ptr = s; // storing decimal ascii value of corresponding
remainder in *ptr
            data = data/base; // Dividing the number(data) by base.
            l++; // incrementing l
       }

        ptr = ptr - l; // decrementing ptr by l to point initial memory
location.
```

```
        uint8_t t; // variable t initialized for swapping data in *ptr.

        uint8_t i=2,j=l; //variables initialized for loop condition.

        while(i<j) // condition for swapping
        {
            t= *(ptr+i);
            *(ptr+i)=*(ptr+j);
            *(ptr+j)=t;
            i++;
            j--;
        }

        //return l; //return length(l) of final ascii string.
    }

    else // condition for invalid base.
    {
        printf("\nError: Invalid parameters\n");
        //return 0;
    }
}
```

6.conversion.h

```
/*************************************************************************
**************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file conversion.h
 * @brief This header file includes data conversion function
declarations.
 * @date October 2, 2017
 *
 * long decription - The conversion.c file includes data conversion
functions for -
 *                      1) integer to ascii string (my_itoa())
 *                      2) ascii string to 32-bit signed integer
(my_atoi())
 *                      3) little endian to big endian
(little_to_big32())
 *                      4) big endian to little endian
(big_to_little32())
 *

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

#ifndef INCLUDES_CONVERSION_H_
#define INCLUDES_CONVERSION_H_

#include <stdio.h>
#include <stdlib.h>
```

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

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

/********************************** my_itoa()
**********************************************************
 *
 * @name  my_itoa(int32_t data,uint8_t* ptr, uint32_t base)
 * @brief function to convert a signed 32-bit integer to an ascii string
and store it at memory loaction
 * @param 1) data - signed 32-bit integer in decimal.
 *        2) *ptr - pointer to a memory location.
 *        3) base - base to which data is converted.
 *
 * long description - This function converts a standard decimal integer
(base 10) to number
 *                    of specified base. The sign of the decimal
number(data)is first determined
 *                    specified memory location (*ptr). The remainder
after taking modulus with base
 *                    and then stored at (using % operator) is converted
to ascii character and
 *                    stored at next memory location (*(ptr + 1)). A
variable 'l' is initialized to
 *                    zero and incremented to determine the length of
string. The decimal number is
 *                    then divided until it becomes zero. After each
modulus operation, ptr is
 *                    incremented and remainder is stored in *ptr. The
characters stored in memory
 *                    locations - (ptr + 1) to (ptr + l) are reversed in
order to store the ascii
 *                    string in correct format. Lastly, the length of the
ascii string (including
 *                    sign) 'l' is returned by the function.
 *
 * @return length of ascii string (uint8_t l)
 *

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

/********************************** my_itoa function declaration
************************************/

void my_itoa(int32_t data,uint8_t* ptr, uint32_t base);


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

```
/********************************** my_atoi()
*************************************************
 *
 * @name  my_atoi(uint8_t* ptr, uint8_t digits, uint32_t base)
 * @brief function to convert an ascii string to a signed 32-bit integer.
 * @param 1) *ptr - pointer to a memory location.
 *        2) digits - length of ascii string(including sign).
 *        3) base - base to which data is converted.
 *
 * long description - This function converts an ascii string (number of
specified base stored
 *                    as an ascii string at the specified memory
location(*ptr)) to a signed
 *                    32-bit integer(int32_t value). The sign of the
number is determined
 *                    by comparing data at ptr location. Data is read
from consecutive memory
 *                    locations (from (ptr+1) to (ptr + digits) and
converted to decimal integer
 *                    ( by multiplying base^n to each number read from
memory location and adding
 *                    them). Finally, the signed decimal integer is
returned by the function.
 *
 * @return signed 32-bit integer in decimal (int32_t value)
 *

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

/******************* my_atoi function declaration
*************************/

int32_t my_atoi(uint8_t *ptr, uint8_t digits, uint32_t base);

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

/********************************** little_to_big32()
*****************************************
 *
 * @name  little_to_big32(uint32_t *data, uint32_t length)
 * @brief function to convert little endian to big endian.
 * @param 1) *data - pointer to a memory location.
 *        2) length - length of ascii string(including sign).
 *
 * long description - This function converts data stored at memory
locations (from (data) to
 *                    (data + length -1)) from little endian to big
endian format for a 32-bit
 *                    word size. Each byte value is stored in uint8_t
variable (a1,a2,a3,a4 to
 *                    break and shift the 32-bit data at memory
location((data + b)). After each
```

```
 *                     conversion memory location is incremented till the
data of all bytes is
 *                     converted.
 *
 * @return 0 if function is executed successfully.
 *

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

/***************************** little_to_big32 function declaration
*******************************/

uint32_t *little_to_big32(uint32_t *data, uint32_t length);

/********************************** big_to_little32()
*****************************************
 *
 * @name  big_to_little32(uint32_t *data, uint32_t length)
 * @brief function to convert big endian to little endian.
 * @param 1) *data - pointer to a memory location.
 *        2) length - length of ascii string(including sign).
 *
 * long description - This function converts data stored at memory
locations (from (data) to
 *                     (data + length -1)) from big endian to little
endian format for a 32-bit
 *                     word size. Each byte value is stored in uint8_t
variable (a1,a2,a3,a4 to
 *                     break and shift the 32-bit data at memory
location((data + b)). After each
 *                     conversion memory location is incremented till the
data of all bytes is
 *                     converted.
 *
 * @return 0 if function is executed successfully.
 *

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

/***************************** big_to_little32 function declaration
*******************************/

uint32_t *big_to_little32(uint32_t *data, uint32_t length);


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

#endif /* INCLUDES_CONVERSION_H_ */
```

7.testcircbuf.c

```
#include "test_circbuf.h"

void test_allocate_free()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;
    CB_init(buff,buff_length);

    assert_ptr_not_equal(buff->buffer,NULL);
    assert_int_equal(buff->length,buff_length);

    CB_destroy(buff);
}

void invalid_pointer()
{
    CB_t *buff=NULL;

    uint8_t buff_length = 10;
    uint8_t ret;

    ret=CB_init(buff,buff_length);

    assert_int_equal(ret,NULL_ERROR);
}

void non_init_buffer()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;
    CB_init(buff,buff_length);


    assert_ptr_not_equal(buff->buffer,NULL);
    assert_int_equal(buff->length,buff_length);
    assert_ptr_equal(buff->buffer,buff->head);
    assert_ptr_equal(buff->buffer,buff->tail);
    assert_int_equal(buff->count,0);

    CB_destroy(buff);
}


void add_remove()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;
```

```
    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t add = 15;
    uint8_t remove = 0;

    CB_buffer_add_item(buff,&add);
    CB_buffer_remove_item(buff,&remove);

    assert_int_equal(remove,add);

    CB_destroy(buff);
}


void buffer_full()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;

    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t i;
    uint8_t ret;

    ret=CB_is_full(buff);
    assert_int_equal(ret,BUFFER_NOT_FULL);

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_add_item(buff,&i);
    }

    ret=CB_is_full(buff);
    assert_int_equal(ret,BUFFER_FULL);

    CB_destroy(buff);
}


void buffer_empty()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;
```

```
    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t i;
    uint8_t ret;

    ret=CB_is_empty(buff);
    assert_int_equal(ret,BUFFER_EMPTY);

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_add_item(buff,&i);
    }

    ret=CB_is_empty(buff);
    assert_int_equal(ret,BUFFER_NOT_EMPTY);

    CB_destroy(buff);
}


void wrap_add()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;

    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t i;
    uint8_t add=12;
    uint8_t remove;
    uint8_t ret;

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_add_item(buff,&i);
    }

    ret=CB_is_full(buff);
    assert_int_equal(ret,BUFFER_FULL);

    CB_buffer_remove_item(buff,&remove);

    CB_buffer_add_item(buff,&add);
    assert_int_equal(*buff->buffer,add);

    CB_destroy(buff);
```

```
}


void wrap_remove()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;

    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t i;
    uint8_t add=12;
    uint8_t remove;
    uint8_t ret;

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_add_item(buff,&i);
    }

    ret=CB_is_full(buff);
    assert_int_equal(ret,BUFFER_FULL);

    CB_buffer_remove_item(buff,&remove);

    CB_buffer_add_item(buff,&add);
    assert_int_equal(*buff->buffer,add);

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_remove_item(buff,&remove);
    }

    assert_ptr_equal(buff->tail,buff->buffer);

    CB_destroy(buff);
}


void overfill()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;

    CB_init(buff,buff_length);

    if(buff->length==buff_length)
```

```
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t i;
    uint8_t add=12;
    uint8_t ret;

    for(i=0;i<buff_length;i++)
    {
      CB_buffer_add_item(buff,&i);
    }

    ret=CB_is_full(buff);
    assert_int_equal(ret,BUFFER_FULL);

    ret=CB_buffer_add_item(buff,&add);
    assert_int_equal(ret,BUFFER_FULL);

    CB_destroy(buff);
}


void overempty()
{
    CB_t b;
    CB_t *buff=&b;

    uint8_t buff_length = 10;

    CB_init(buff,buff_length);

    if(buff->length==buff_length)
      assert_ptr_not_equal(buff->buffer,NULL);

    uint8_t remove;
    uint8_t ret;

    ret=CB_buffer_remove_item(buff,&remove);
    assert_int_equal(ret,BUFFER_EMPTY);

    CB_destroy(buff);
}

8.test_memory.c

#include "test_memory.h"

void test_memmove1()
{
   uint8_t * ret;
   uint8_t * ptra=NULL;
   uint8_t * ptrb=NULL;

   ret=my_memmove(ptra, ptrb, 15);
```

```
   assert_ptr_equal(ret,NULL);
}


void test_memmove2()
{
   uint8_t i;
   uint8_t * set;
   uint8_t * ptra;
   uint8_t * ptrb;
   set = (uint8_t*) malloc(32*sizeof(uint8_t));

   ptra = &set[0];
   ptrb = &set[16];

   for( i = 0; i < 32; i++)
   {
      set[i] = i;
   }

   my_memmove(ptra, ptrb, 16);

   for (i = 0; i < 16; i++)
   {
        assert_int_equal(*(ptrb+i),*(ptra+i));

   }

}


void test_memmove3()
{
   uint8_t i;
   uint8_t * set;
   uint8_t * ptra;
   uint8_t * ptrb;
   set = (uint8_t*) malloc(32*sizeof(uint8_t));

   ptra = &set[8];
   ptrb = &set[0];

   for( i = 0; i < 32; i++)
   {
      set[i] = i;
   }

   my_memmove(ptra, ptrb, 16);

   for (i = 0; i < 16; i++)
   {
        assert_int_equal(*(set+i),(i+8));
   }
}
```

```
void test_memmove4()
{
   uint8_t i;
   uint8_t * set;
   uint8_t * ptra;
   uint8_t * ptrb;
   set = (uint8_t*) malloc(32*sizeof(uint8_t));

   ptra = &set[0];
   ptrb = &set[8];

   for( i = 0; i < 32; i++)
   {
      set[i] = i;
   }

   my_memmove(ptra, ptrb, 16);

   for (i = 0; i < 16; i++)
   {
      assert_int_equal(*(set+i+8),i);

   }

}

void test_memset1()
{
   uint8_t *set=NULL;
   uint8_t *ret;

   ret=my_memset(set,5,3);

   assert_ptr_equal(ret,NULL);

}

void test_memset2()
{
   uint8_t i;
   uint8_t *set;

   set = (uint8_t*) malloc(40*sizeof(uint8_t));

   my_memset(set,5,3);
   for(i=0;i<5;i++)
   {
      assert_int_equal(*(set+i),3);
   }
}

void test_memzero1()
```

```
{
   uint8_t *set=NULL;
   uint8_t *ret;

   ret=my_memzero(set,5);

   assert_ptr_equal(ret,NULL);

}

void test_memzero2()
{
   uint8_t i;
   uint8_t *set;

   set = (uint8_t*) malloc(40*sizeof(uint8_t));

   my_memzero(set,5);
   for(i=0;i<5;i++)
   {
      assert_int_equal(*(set+i),0);
   }
}

void test_memreverse1()
{
   //uint8_t i;
   //uint8_t set[8] = {0x3F, 0x73, 0x72, 0x33, 0x54, 0x43, 0x72, 0x26};
   uint8_t *rev = NULL;
   uint8_t *ret;

   ret=my_reverse(rev,8);

   assert_ptr_equal(ret,NULL);
}

void test_memreverse2()
{
   uint8_t i;
   uint8_t set[7] = {0x3F, 0x73, 0x72, 0x33, 0x54, 0x43, 0x72};
   uint8_t temp[7];
   for(i=0;i<7;i++)
   {
      temp[i]=set[i];
   }

   uint8_t *rev = set;
   uint8_t *ret;

   ret=my_reverse(rev,7);

   for(i=0;i<7;i++)
   {
      assert_int_equal(*(ret+i),temp[6-i]);
```

```
    }
}

void test_memreverse3()
{
    uint8_t i;
    uint8_t set[8] = {0x3F, 0x73, 0x72, 0x33, 0x54, 0x43, 0x72, 0x26};
    uint8_t temp[8];
    for(i=0;i<8;i++)
    {
        temp[i]=set[i];
    }

    uint8_t *rev = set;
    uint8_t *ret;

    ret=my_reverse(rev,8);

    for(i=0;i<8;i++)
    {
        assert_int_equal(*(ret+i),temp[7-i]);
    }
}

void test_memreverse4()
{
    uint8_t i;
    uint8_t set[255];
    uint8_t temp[255];
    for(i=0;i<255;i++)
    {
        temp[i]=i+1;
        set[i]=i+1;
    }

    uint8_t *rev = set;
    uint8_t *ret;

    ret=my_reverse(rev,255);

    for(i=0;i<255;i++)
    {
        assert_int_equal(*(ret+i),temp[254-i]);
    }
}
```

9.test_conversion.c

```
#include "test_conversion.h"


void test_big_to_little1()
{
    uint32_t *data = NULL;
```

```
   uint32_t *ret;

   ret=big_to_little32(data, 1);

   assert_ptr_equal(ret,NULL);
}

void test_big_to_little2()
{
   uint32_t a=0x12345678;
   uint32_t *data = &a;
   uint32_t *ret;

   ret=big_to_little32(data, 1);

   assert_int_equal(*ret,0x78563412);
}


void test_little_to_big1()
{
   uint32_t *data = NULL;
   uint32_t *ret;

   ret=little_to_big32(data, 1);

   assert_ptr_equal(ret,NULL);
}

void test_little_to_big2()
{
   uint32_t a=0x12345678;
   uint32_t *data = &a;
   uint32_t *ret;

   ret=little_to_big32(data, 1);

   assert_int_equal(*ret,0x78563412);
}
```

10.test.c

```
/*************************************************************************
*************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @file test.c
 * @brief This file includes cmocka test functions
 * @date October 24, 2017
 *
 * long description - The uart.c file includes functions to -
 *                      1) test memory.c
 *                      2) test conversion.c
```

```
 *                      3) test cirbuf.c

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


#include "test_memory.h"
#include "test_circbuf.h"
#include "test_conversion.h"


int main()
{
  const struct CMUnitTest tests[] = {
    cmocka_unit_test(test_memmove1),
    cmocka_unit_test(test_memmove2),
    cmocka_unit_test(test_memmove3),
    cmocka_unit_test(test_memmove4),
    cmocka_unit_test(test_memset1),
    cmocka_unit_test(test_memset2),
    cmocka_unit_test(test_memzero1),
    cmocka_unit_test(test_memzero2),
    cmocka_unit_test(test_memreverse1),
    cmocka_unit_test(test_memreverse2),
    cmocka_unit_test(test_memreverse3),
    cmocka_unit_test(test_memreverse4),
    cmocka_unit_test(test_allocate_free),
    cmocka_unit_test(invalid_pointer),
    cmocka_unit_test(non_init_buffer),
    cmocka_unit_test(add_remove),
    cmocka_unit_test(buffer_full),
    cmocka_unit_test(buffer_empty),
    cmocka_unit_test(wrap_add),
    cmocka_unit_test(wrap_remove),
    cmocka_unit_test(overfill),
    cmocka_unit_test(overempty),
    cmocka_unit_test(test_big_to_little1),
    cmocka_unit_test(test_big_to_little2),
    cmocka_unit_test(test_little_to_big1),
    cmocka_unit_test(test_little_to_big2),
  };

  return cmocka_run_group_tests(tests, NULL, NULL);
}


11.test_cirbuf.h


#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
```

```
#include "circbuf.h"


void test_allocate_free();
void invalid_pointer();
void non_init_buffer();
void add_remove();
void buffer_full();
void buffer_empty();
void wrap_add();
void wrap_remove();
void overfill();
void overempty();
```

12.test_memory.h

```
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include "memory.h"

void test_memmove1();
void test_memmove2();
void test_memmove3();
void test_memmove4();

void test_memset1();
void test_memset2();

void test_memzero1();
void test_memzero2();

void test_memreverse1();
void test_memreverse2();
void test_memreverse3();
void test_memreverse4();
```

13.test_conversion.h

```
#include <stdlib.h>
#include <stdarg.h>
#include <stddef.h>
#include <setjmp.h>
#include <cmocka.h>
#include "conversion.h"

void test_big_to_little1();
void test_big_to_little2();
void test_little_to_big1();
void test_little_to_big2();
```

14.uart.h

```
/*****************************************************************************
*************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file uart.h
 * @brief This header file includes UART functions
 * @date October 24, 2017
 *
 * long description - The uart.h file includes function declarations to -
 *                       1) configure UART (UART_Configure())
 *                       2) send a UART character (UART_send())
 *                       3) send n UART characters (UART_send_n())
 *                       4) receive UART character (UART_receive())
 *                       5) receive n UART characters (UART_receive_n())
 *                       6) handle UART interrupts (UART0_IRQHandler())
 *
 *
*****************************************************************************
*************************/


#ifndef INCLUDES_UART_H_
#define INCLUDES_UART_H_

/*----- Header-Files ------------------------------------------------------
---*/

#include "MKL25Z4.h"
#include "circbuf.h"
#include "conversion.h"

/*----- Extern Declarations -----------------------------------------------
----------*/

extern uint8_t Rx_Data;
extern uint8_t Tx_Data;
extern CB_t Tx_Buffer;
extern CB_t Rx_Buffer;

/*----- Macros for baud rate ----------------------------------------------
-----------*/

#define BAUD 115200
#define BAUD_RATE ((SystemCoreClock)/((BAUD)*(16)))

/********************************** UART_configure()
*************************************************
 *
 * @name   -  UART_configure()
 * @brief  -  function to initialize UART
 * @param  -  none
```

```
 *
 * long description - This function configures UART control and status
registers, sets baud rate, enables
 *                              interrupts, selects clock mode and sets
oversampling ratio.
 *
 * @return -  SUCCESS
 *

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

/******************************** UART_configure() function declaration
**********************************/

CB_status UART_configure();

/********************************** UART_send()
***************************************************
 *
 * @name   -  UART_send()
 * @brief  -  function to send a character
 * @param  -  *data0 - pointer to data
 *
 * long description - This function sends a character by writing data to
UART0 data register
 *
 * @return -  TX_SUCCESS
 *

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

/******************************** UART_send() function declaration
**********************************/

CB_status UART_send(uint8_t *data0);

/********************************** UART_send_n()
***************************************************
 *
 * @name   -  UART_send_n()
 * @brief  -  function to send n characters
 * @param  -  *data0 : pointer to data
 * @param  -  length : length of data bytes
 *
 * long description - This function sends n characters by writing data to
UART0 data register
 *
 * @return -  TX_SUCCESS
 *

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

```
/******************************** UART_send_n() function declaration
**********************************/

CB_status UART_send_n(uint8_t *data0, uint8_t length);

/********************************** UART_receive()
*************************************************
 *
 * @name   -  UART_receive()
 * @brief  -  function to receive a character
 * @param  -  *data0 - pointer to data
 *
 * long description - This function receives a character by reading data
from UART0 data register
 *
 * @return -  TX_SUCCESS
 *

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

/******************************** UART_receive() function declaration
**********************************/

CB_status UART_receive(uint8_t *data0);

/********************************** UART0_IRQHandler()
*************************************************
 *
 * @name   -  UART0_IRQHandler()
 * @brief  -  function to handle UART0 interrupts
 * @param  -  none
 *
 * long description - This function handles UART0 interrupts while
receive and transmit operations.
 *
 * @return -  TX_IRQ : transmit interrupt
 * @return -  RX_IRQ : receive interrupt
 *

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

/******************************** UART0_IRQHandler() function declaration
**********************************/

CB_status UART_receive_n(uint8_t *data0, uint8_t length);

#endif /* INCLUDES_UART_H_ */
```

15.project2.h

```
/*************************************************************************
*************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file project2.h
 * @brief This header file includes UART data sorting and analysis
function declarations.
 * @date October 25, 2017
 *
 * long description - The project2.c file includes functions declarations
of functions to -
 *                       1) perform UART data sorting and
analysis(project_2())
 *                       2) Transmit data analysis in a tabular
format(Table_Stats())
 *

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

#ifndef INCLUDES_PROJECT2_H_
#define INCLUDES_PROJECT2_H_

/*----- Macros for character strings ----------------------------------
-------------------*/

#define NUM "Number of Numeric Characters       : "
#define ALPHA "Number of Alphabetic Characters    : "
#define PUNC "Number of Punctuation Characters   : "
#define MISC "Number of Miscellaneous Characters : "

/*----- Extern Declarations -------------------------------------------
----------*/

extern uint8_t char_count[4];
extern uint8_t Rx_Data;
extern uint8_t Tx_Data;


extern CB_t Tx_Buffer;
extern CB_t Rx_Buffer;


void Table_Stats_1();
/********************************** Table_Stats()
*************************************************
 *
 * @name   -  Table_Stats()
 * @brief  -  function to transmit analyzed data result in a tabular
format.
 * @param  -  none
 *
 * long description - This function transmits data results obtained after
analyzing and sorting of
```

```
 *                              received UART characters (data).
 *
 * @return -  void
 *

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

/******************************** Table_Stats() function declaration
**********************************/


void Table_Stats();
/********************************** project_2()
*************************************************
 *
 * @name   -  project_2()
 * @brief  -  function to analyze and sort received data.
 * @param  -  none
 *
 * long description - This function removes received data from receive
buffer and analyzes it to
 *                              increment count of the respective
character type. On receiving Tab character,
 *                              Table_Stats() function is called to send
analyzed data results in a tabular
 *                              format.
 *
 * @return -  void
 *

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

/********************************** project_2() function definition
**********************************/

void project_2();

#endif /* INCLUDES_PROJECT2_H_ */
```

16.uart.c

```
/*******************************************************************************
*************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file uart.c
 * @brief This file includes UART functions
 * @date October 24, 2017
 *
 * long description - The uart.c file includes functions to -
 *                      1) configure UART (UART_Configure())
 *                      2) send a UART character (UART_send())
```

```
 *                       3) send n UART characters (UART_send_n())
 *                       4) receive UART character (UART_receive())
 *                       5) receive n UART characters (UART_receive_n())
 *                       6) handle UART interrupts (UART0_IRQHandler())
 *
 *

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


/*----- Header-Files ---------------------------------------------------------
---*/

#include "uart.h"
#include "circbuf.h"
#include "project2.h"

/********************************** UART_configure()
*****************************************************
 *
 * @name   -  UART_configure()
 * @brief  -  function to initialize UART
 * @param  -  none
 *
 * long description - This function configures UART control and status
registers, sets baud rate, enables
 *                              interrupts, selects clock mode and sets
oversampling ratio.
 *
 * @return -  SUCCESS
 *

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

/******************************** UART_configure() function definition
*********************************/


CB_status UART_configure()
{
      SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
      SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;

      PORTA_PCR1 |= PORT_PCR_MUX(2);
      PORTA_PCR2 |= PORT_PCR_MUX(2);

      SIM->SOPT2 &= ~(SIM_SOPT2_PLLFLLSEL_MASK);
      SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL(1);
      SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);

      uint16_t baud_rate = BAUD_RATE;
      UART0_BDL = (uint8_t)(baud_rate & UART0_BDL_SBR_MASK) ;
```

```
	UART0_BDH = (uint8_t)((baud_rate>>8) & UART0_BDH_SBR_MASK);

	UART0_C1 = 0x00;
	UART0_C3 = 0x00;
	UART0_S1 = 0x00;
	UART0_C4 = UART0_C4_OSR(15);

	UART0_C2 &= ~(UART0_C2_RE_MASK | UART0_C2_TE_MASK);
	UART0_C2 |=  (UART0_C2_RE_MASK | UART0_C2_TE_MASK);
	UART0_C2 |= UART0_C2_RIE_MASK;

	NVIC_EnableIRQ(UART0_IRQn);

	return SUCCESS;
}

/********************************* UART_send()
*************************************************
 *
 * @name   -  UART_send()
 * @brief  -  function to send a character
 * @param  -  *data0 - pointer to data
 *
 * long description - This function sends a character by writing data to
UART0 data register
 *
 * @return -  TX_SUCCESS
 *

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

/********************************* UART_send() function definition
**********************************/

CB_status UART_send(uint8_t *data0)
{
	UART0_D = *data0;
	return SUCCESS;
}

/********************************* UART_send_n()
*************************************************
 *
 * @name   -  UART_send_n()
 * @brief  -  function to send n characters
 * @param  -  *data0 : pointer to data
 * @param  -  length : length of data bytes
 *
 * long description - This function sends n characters by writing data to
UART0 data register
 *
 * @return -  TX_SUCCESS
 *
```

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

/******************************** UART_send_n() function definition
**********************************/


CB_status UART_send_n(uint8_t *data0, uint8_t length)
{
      uint8_t i = 0;

      for(i=0;i<length;i++)
      {
            //while(((UART0_S1)&(UART0_S1_TDRE_MASK)) !=
(UART0_S1_TDRE_MASK));
            UART0_D = *(data0 + i);
      }
      return SUCCESS;
}

/********************************** UART_receive()
*************************************************
 *
 * @name   -  UART_receive()
 * @brief  -  function to receive a character
 * @param  -  *data0 - pointer to data
 *
 * long description - This function receives a character by reading data
from UART0 data register
 *
 * @return -  TX_SUCCESS
 *

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

/******************************** UART_receive() function definition
**********************************/


CB_status UART_receive(uint8_t *data0)
{
      *(data0) = UART0_D;
      return RX_SUCCESS;
}

/********************************** UART_receive_n()
*************************************************
 *
 * @name   -  UART_receive_n()
 * @brief  -  function to receive n characters
 * @param  -  *data0 : pointer to data
 * @param  -  length : length of data bytes received
```

```
 *
 * long description - This function receives n characters by reading data
from UART0 data register
 *
 * @return -  TX_SUCCESS
 *
**************************************************************************
***************************/

/******************************** UART_receive_n() function definition
**********************************/

CB_status UART_receive_n(uint8_t *data0, uint8_t length)
{
      uint8_t i = 0;

            for(i=0;i<length;i++)
            {
                  *(data0 + length) = UART0_D;
            }

      return RX_SUCCESS;
}

/********************************** UART0_IRQHandler()
*****************************************************
 *
 * @name   -  UART0_IRQHandler()
 * @brief  -  function to handle UART0 interrupts
 * @param  -  none
 *
 * long description - This function handles UART0 interrupts while
receive and transmit operations.
 *
 * @return -  TX_IRQ : transmit interrupt
 * @return -  RX_IRQ : receive interrupt
 *
**************************************************************************
**********************************/

/******************************** UART0_IRQHandler() function definition
**********************************/

CB_status UART0_IRQHandler()
{

      if((UART0_S1 & UART0_S1_TDRE_MASK) && (!CB_is_empty(&Tx_Buffer))){
            CB_buffer_remove_item(&Tx_Buffer, &Tx_Data);
            UART_send(&Tx_Data);
            if(CB_is_empty(&Tx_Buffer)){
                  UART0_C2 &= ~UART0_C2_TIE_MASK;
            }
```

```
            return TX_IRQ;
      }

      if((UART0_S1 & UART0_S1_RDRF_MASK ) && (!CB_is_full(&Rx_Buffer))){
            UART_receive(&Rx_Data);
            CB_buffer_add_item(&Rx_Buffer, &Rx_Data);
            if(CB_is_full(&Rx_Buffer)){
                  UART0_C2 &= ~UART0_C2_RIE_MASK;
            }
            return RX_IRQ;
      }

      return NULL_ERROR;
}
```

17.project2.c

```
/*************************************************************************
**************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file project2.c
 * @brief This file includes UART data sorting and analysis functions
 * @date October 25, 2017
 *
 * long description - The project2.c file includes functions to -
 *                      1) perform UART data sorting and
analysis(project_2())
 *                      2) Transmit data analysis in a tabular
format(Table_Stats())
 *
*************************************************************************
*************************/


/*----- Header-Files ------------------------------------------------------
---*/

#include "uart.h"
#include "conversion.h"
#include "project2.h"

/*----- Buffer Initialization ---------------------- ------------------
---*/

CB_t Tx_Buffer;
CB_t Rx_Buffer;

/*----- Global Variables ---------------------------------- ----------
---*/

uint8_t Rx_Data;
```

```
uint8_t Tx_Data;
uint8_t char_count[4];

/********************************* Table_Stats()
*************************************************
 *
 * @name   -  Table_Stats()
 * @brief  -  function to transmit analyzed data result in a tabular
format.
 * @param  -  none
 *
 * long description - This function transmits data results obtained after
analyzing and sorting of
 *                              received UART characters (data).
 *
 * @return -  void
 *

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

/******************************** Table_Stats() function definition
*********************************/

void Table_Stats()
{
      /*----- Variable and Array Declaration ------------------------ ---
------------------*/

    uint8_t char_str[4][41]={NUM,ALPHA,PUNC,MISC};
    uint8_t l;
    uint8_t k;



    /*-----  ------------------------ ----------------------*/

    for(k=0;k<4;k++)
    {
      for(l=0;l<41;l++)
      {
            CB_buffer_add_item(&Tx_Buffer,&char_str[k][l]);
      }

      uint8_t buff[4];
      my_itoa(char_count[k],buff,10);

      CB_buffer_add_item(&Tx_Buffer, &buff[1]);
      CB_buffer_add_item(&Tx_Buffer, &buff[2]);
      char_count[k] = 0;

      for(l=0;l<4;l++){
            buff[l] = 0;
      }
```

```
      uint8_t line = 0x0d;
      CB_buffer_add_item(&Tx_Buffer, &line);
      line = 0x0a;
      CB_buffer_add_item(&Tx_Buffer, &line);
    }

    /*----- Enabling Transmit Interrupt -------------------------- -----
----------------*/

    UART0_C2 &=~(UART0_C2_TIE_MASK); /* Clear TIE bit field */
    UART0_C2 |= UART0_C2_TIE_MASK;   /*Set TIE bit field */
}

void project_2()
{

    SystemInit();

    UART_configure();

    CB_init(&Tx_Buffer,255);
    CB_init(&Rx_Buffer,255);


    uint8_t Rxd_Data;

    while(1)
    {
      CB_buffer_remove_item(&Rx_Buffer, &Rxd_Data);

            if((Rxd_Data>47) && (Rxd_Data<58))
            {
                char_count[0]++;
                Rxd_Data=0;
            }

            else if(((Rxd_Data>64) && (Rxd_Data<91))||((Rxd_Data>96) &&
(Rxd_Data<123)))
            {
                 char_count[1]++;
                Rxd_Data=0;
            }

            else if(((Rxd_Data>32) && (Rxd_Data<48))||((Rxd_Data>57) &&
(Rxd_Data<65))||((Rxd_Data>90) && (Rxd_Data<97))||((Rxd_Data>122) &&
(Rxd_Data<127)))
            {
                 char_count[2]++;
                Rxd_Data=0;
            }

            else if(Rxd_Data==9)
            {
```

```
                Table_Stats();
                Rxd_Data=0;
            }

            else
if(((Rxd_Data>90)&&(Rxd_Data<97))||(Rxd_Data==127)||(Rxd_Data==27))
            {
                char_count[3]++;
               Rxd_Data=0;
            }


       }


    }
```

18.main.c

```
#include "uart.h"
#include "project2.h"


int main(void)
{
#ifdef PROJECT2
     project_2();
#endif
}
```

19.MAKEFILE

```
-include sources.mk


ifeq ($(PLATFORM),HOST)
CC=gcc
CFLAGS= -Wall -Werror -g -O0 -std=c99 -DPROJECT2
SOURCES=$(HOST_SRCS_C)
INCLUDES=$(INCLUDE_HOST) $(INCLUDE_ARM)

else ifeq ($(PLATFORM),KL25Z)
CC=arm-none-eabi-gcc
CFLAGS= -Wall -Werror -g -O0 -std=c99 \
-mcpu=cortex-m0plus -march=armv6-m -mthumb \
-mfloat-abi=soft -mfpu=fpv4-sp-d16 -specs=nosys.specs -DPROJECT2
SOURCES=$(HOST_SRCS_C) $(KL25Z_SRCS_C) $(KL25Z_SRCS_S)
INCLUDES=$(INCLUDE_HOST) $(INCLUDE_ARM) $(INCLUDE_KL25Z)
LDFLAGS=-T ../platform/MKL25Z128xxx4_flash.ld
```

```
else ifeq ($(PLATFORM),BBB)
CC=arm-linux-gnueabihf-gcc
CFLAGS=-Wall -Werror -g -O0 -std=c99 -mcpu=cortex-a8 -mthumb -mfloat-
abi=hard -DPROJECT2
SOURCES=$(HOST_SRCS_C)
INCLUDES=$(INCLUDE_HOST) $(INCLUDE_ARM)

else ifeq($(PLATFORM),TEST)
CC=gcc
CFLAGS= -Wall -Werror -g -O0 -std=c99
INCLUDES=$(INCLUDE_HOST) $(INCLUDE_ARM)
endif


ifeq ($(PLATFORM),KL25Z)
OBJECTS:= $(HOST_SRCS_C:.c=.o) $(KL25Z_SRCS_C:.c=.o)
$(KL25Z_SRCS_S:.S=.o)
DEPENDANCY:= $(HOST_SRCS_C:.c=.d) $(KL25Z_SRCS_C:.c=.d)
$(KL25Z_SRCS_S:.S=.d)

else ifeq ($(PLATFORM),BBB)
OBJECTS:= $(SOURCES:.c=.o)
DEPENDANCY:= $(SOURCES:.c=.d)

else ifeq ($(PLATFORM),HOST)
OBJECTS:= $(SOURCES:.c=.o)
DEPENDANCY:= $(SOURCES:.c=.d)

endif


.PHONY: compile-all build exit

# Targets

%.i: %.c
	-@echo ' '
	-@echo 'preprocessor file .i created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -E -o $@ $<

%.asm: %.c
	-@echo ' '
	-@echo 'assembly file created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -S -o $@ $<

%.o: %.c
	-@echo ' '
	-@echo 'object file for .c file created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -c -o $@ $<

%.o: %.S
	-@echo ' '
	-@echo 'object file for .s file created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -c -o $@ $<
```

```
%.d: %.c
	-@echo ' '
	-@echo 'dependency file for .c file.. created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -M -o $@ $<

%.d: %.S
	-@echo ' '
	-@echo 'dependency file for .S file.. created: '
	-$(CC) $(CFLAGS) $(INCLUDES) -M -o $@ $<

# compile but not link

compile-all:$(OBJECTS)


# compile and link

build:$(OBJECTS) $(DEPENDANCY)
	-@echo ' '
	-@echo 'build in process: '
	-$(CC) $(CFLAGS) $(INCLUDES) $(LDFLAGS) -Xlinker -Map=project1.map
$(OBJECTS) -o project1.elf
	-@echo ' '
	-size project1.elf
	-@echo ' '
	-ls -sh project1.elf

# library for unit test

TEST_SRCS = \
	memory.c \
	conversion.c \
	circbuf.c \
	test_memory.c \
	test_conversion.c \
	test_circbuf.c

TEST_OBJS = $(TEST_SRCS:%.c=%.o)

TEST_HEADERS = $(TEST_SRCS:%.c=%.h)

LIB = libutils.a


# test
MAIN_SRCS = \
	test.c

MAIN_OBJS = $(MAIN_SRCS:%.c=%.o)

MAIN_EXE = test.out
```

```
.PHONY : test clean

# build and run cmocka test

test : $(MAIN_EXE)
	./$(MAIN_EXE)

$(LIB) : $(TEST_OBJS)
	$(AR) $(ARFLAGS) $@ $^

$(MAIN_EXE) : $(MAIN_SRCS) $(LIB)
	$(CC) $(CFLAGS) $(INCLUDES) -o $@ $^ -lcmocka

# delete created files

clean :
	-@echo ' '
	-@echo 'clean all the generated files: '
	-rm -rf *~ *.o $(LIB) $(EXE) $(MAIN_EXE) *.dSYM/
	-$(RM) *.i *.asm *.o *.d *.out project1.map project1.elf
	-@echo ' '



###############################################################################


20. sources.mk


# Add Source files to build variables

HOST_SRCS_C = ./main.c ./project2.c ./conversion.c ./memory.c ./uart.c
./circbuf.c

KL25Z_SRCS_C = ./system_MKL25Z4.c

KL25Z_SRCS_S = ./startup_MKL25Z4.S



# Define Platform specific include files

INCLUDE_HOST= -I ../include/common

INCLUDE_ARM= -I ../include/CMSIS

INCLUDE_KL25Z= -I ../include/kl25z
```