1) main.c

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file main.c
 * @brief This file includes logging functions
 * @date December 3, 2017
 *
 * long description-1) main file of project 3 to perform logger,
 *                     profiling spi_nrf testing for kl25z
 *                   2) logger for BBB and Host
 *

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

#ifdef PROJECT3
#include "project3.h"
#endif

#ifdef PROJECT3_1
#include "project3_1.h"
#endif

int main(void)
{
      #ifdef PROJECT3
      project3();
      #endif
      #ifdef PROJECT3_1
      project3_1();
      #endif
}
```

2) project3.c

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file project3.c
 * @brief This file includes logging functions
 * @date December 3, 2017
 *
 * long description - The logger.c file includes functions to -
 *                      1) project3() to initialise logging functions
 *                      2) data() to log data receive and data
analysis
```

```
 *                       3) profiling() to perform profiling function
on library, NON-DMA and DMA
 *                          memmove and memmset functions
 *

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

#include "project3.h"


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

uint8_t Rx_Data;
uint8_t Tx_Data;
uint8_t Rxd_Data;

uint8_t character_count[4]={0,0,0,0};
uint8_t char_str[4][10];
uint8_t *num = &char_str[0][0];
uint8_t *alpha = &char_str[1][0];
uint8_t *punc = &char_str[2][0];
uint8_t *misc = &char_str[3][0];
uint8_t time[4];
uint32_t start_time;
uint32_t end_time;
uint8_t t;


/********************************** project3()
*********************************************************
 *
 * @name   -  project3()
 * @brief  -  function to pass logs to log buffer and then print it to
terminal through terminal
 *
 * long description - This function takes data input from user and
depending on the received data
 *                                 it performs various logging functions
such as data analysis and profiling
 *
 * @return -  void
 *

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

/********************************** project3() function definition
*********************************/

void project3()
```

```
{
	SystemInit();
	//Initialize System
	int i;
	for(i=0;i<10;i++)
	{
		*(num+i)=0;
		*(alpha+i)=0;
		*(punc+i)=0;
		*(misc+i)=0;
	}
	log_mode = NON_BLOCKING_MODE;			//selest
blocking or non-blocking mode for logger

	CB_init(&Rx_Buffer,50);
	//Initialize Receive buffer

	RTC_init();
	//Initialize RTC
	UART_configure();
	//Initialize UART

	while(1)
	{
		CB_buffer_remove_item(&Rx_Buffer, &Rxd_Data);
		if(Rxd_Data==61)				//if received
data is '=' then initialize LOG
		{
			Log_buffer_init(&log_buffer,500);	//initialize log
buffer to transmit data to terminal through UART
			Rxd_Data=0;
			LOG(LOG_INITIALIZED,NULL);		//log LOG
INITIALIZED to terminal
			LOG(SYSTEM_INITIALIZED,NULL);		//log SYSTEM
INITIALIZED to terminal
			GPIO_nrf_init();			//Initialize
GPIO
			RTC_IER |= RTC_IER_TSIE(1);		//Seconds
interrupt enable

			while(1)
			{
				CB_buffer_remove_item(&Rx_Buffer, &Rxd_Data);
				if(Rxd_Data==47)		//if received
data is '/' then initialize DATA INPUT
				{
					RTC_IER &= ~RTC_IER_TSIE(1);	//Seconds
interrupt disable
					Rxd_Data=0;
					LOG(DATA_INPUT_ENABLED,NULL);	//log DATA
INPUT ENABLED to terminal
					data();
```

```
                              RTC_IER |= RTC_IER_TSIE(1);      //Seconds
interrupt enable
                        }

                        if(Rxd_Data==46)                 //if received
data is '.' then initialize PROFILING
                        {
                              RTC_IER &= ~RTC_IER_TSIE(1);     //Seconds
interrupt disable
                              Rxd_Data=0;
                              LOG(PROFILING_STARTED,NULL);     //log
PROFILING STARTED to terminal
                              profiling();
                              RTC_IER |= RTC_IER_TSIE(1);      //Seconds
interrupt enable
                        }

                        if(Rxd_Data==61)                 //if received
data is '=' then STOP LOGGING
                        {
                              RTC_IER &= ~RTC_IER_TSIE(1);     //Seconds
interrupt disable
                              Rxd_Data=0;
                              break;
                        }
                  }

                  LOG(SYSTEM_HALTED,NULL);               //log SYSTEM
HALTED to terminal
                  Log_buffer_destroy(&log_buffer);
            }

            if(Rxd_Data==44)                             //if received
data is ',' then show results of profiling
            {
                  profile_test();
            }

            if(Rxd_Data==39)                             //if received
data is ''' then show results of SPI_NRF
            {
                  spi_nrf_test();
            }
      }
}

/********************************** data()
*********************************************************
 *
 * @name   -  data()
 * @brief  -  function to perform data analysis on the data received
through uart
```

```
 *
 * @return -  void
 *

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

/********************************** data() function definition
*********************************/

void data()
{
      while(1)
      {

      //----- Removing Data from Rx Buffer  ---------------------------
----------------- ------

           CB_buffer_remove_item(&Rx_Buffer, &Rxd_Data);

           //----- Check for Numeric characters  ---------------------
----------------------- ------
           if((Rxd_Data>47) && (Rxd_Data<58))
          {
                LOG(DATA_RECEIVED,NULL);              //log DATA
RECEIVED to terminal
                character_count[0]++;
                *num = Rxd_Data;
                num++;
                Rxd_Data=0;
          }

          //----- Check for Alphabetic characters  --------------------
--------------------- ------

           else if(((Rxd_Data>64) && (Rxd_Data<91))||((Rxd_Data>96) &&
(Rxd_Data<123)))
          {
                LOG(DATA_RECEIVED,NULL);              //log DATA
RECEIVED to terminal
                character_count[1]++;
                *alpha = Rxd_Data;
                alpha++;
                Rxd_Data=0;
          }

          //----- Check for Punctuation characters  -------------------
---------------------- ----

           else if(((Rxd_Data>32) && (Rxd_Data<46))||((Rxd_Data>57) &&
(Rxd_Data<61))||((Rxd_Data>90) && (Rxd_Data<97))||((Rxd_Data>122) &&
(Rxd_Data<127)))
```

```
           {
                LOG(DATA_RECEIVED,NULL);              //log DATA
RECEIVED to terminal
                character_count[2]++;
                *punc = Rxd_Data;
                punc++;
                Rxd_Data=0;
           }

          //----- Check for Tab character - ---------------------------
---------------------- ------

           else if(Rxd_Data==9)                       //if received
data is 'TAB' then start data analysis
          {
                num = &char_str[0][0];
                alpha = &char_str[1][0];
                punc = &char_str[2][0];
                misc = &char_str[3][0];
                LOG(DATA_ANALYSIS_STARTED,NULL); //log DATA ANALYSIS
STARTED to terminal
                LOG(DATA_ALPHA_COUNT,NULL);           //log DATA ALPHA
COUNT to terminal
                LOG(DATA_NUMERIC_COUNT,NULL);         //log DATA
NUMERIC COUNT to terminal
                LOG(DATA_PUNCTUATION_COUNT,NULL);     //log DATA
PUNCTUATION COUNT to terminal
                LOG(DATA_MISC_COUNT,NULL);            //log DATA MISC
COUNT to terminal
                character_count[0]=0;
                character_count[1]=0;
                character_count[2]=0;
                character_count[3]=0;
                LOG(DATA_ANALYSIS_COMPLETED,NULL);    //log DATA
ANALYSIS COMPLETED to terminal
              Rxd_Data=0;
          }

           //----- Check for Miscellaneous characters  ---------------
------------------------------

           else
if(((Rxd_Data>90)&&(Rxd_Data<97))||(Rxd_Data==127)||(Rxd_Data==27))
          {
                LOG(DATA_RECEIVED,NULL);              //log DATA
RECEIVED to terminal
                character_count[3]++;
                *misc = Rxd_Data;
                misc++;
              Rxd_Data=0;
          }
```

```
           if(Rxd_Data==47)                              //if received
data is '/' then disable input
           {
                 Rxd_Data=0;
                 LOG(DATA_INPUT_DISABLED,NULL);
                 break;
           }
     }
}

/********************************** profiling()
*************************************************************
 *
 * @name   -  profiling()
 * @brief  -  function to perform profiling operation on library, NON-
DMA and DMA memmove and memset functions
 *
 * @return -  void
 *

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

/********************************** profiling() function definition
**********************************/
void profiling()
{
     while(1)
     {
           CB_buffer_remove_item(&Rx_Buffer, &Rxd_Data);
           if(Rxd_Data==49)                              //if received
data is '1' then start profiling for library memmove function
           {
                 Rxd_Data=0;
                 lib_memmove_test(10);
                 LOG(INFO,"lib-memmove for 10 bytes");//log INFO to
terminal
           }

           if(Rxd_Data==50)                              //if received
data is '2' then start profiling for library memset function
           {
                 Rxd_Data=0;
                 lib_memset_test(10);
                 LOG(INFO,"lib-memset for 10 bytes");//log INFO to
terminal
           }

           if(Rxd_Data==51)                              //if received
data is '3' then start profiling for NON-DMA memmove function
           {
                 Rxd_Data=0;
```

```
                my_memmove_test(10);
                LOG(INFO,"my-memmove for 10 bytes");//log INFO to
terminal
           }


           if(Rxd_Data==52)                              //if received
data is '4' then start profiling for NON-DMA memset function
           {
                Rxd_Data=0;
                my_memset_test(10);
                LOG(INFO,"my-memset for 10 bytes");   //log INFO to
terminal
           }

           if(Rxd_Data==53)                              //if received
data is '5' then start profiling for DMA memmove function
           {
                Rxd_Data=0;
                dma_memmove_test(10);
                LOG(INFO,"dma-memmove for 10 bytes");//log INFO to
terminal
           }

           if(Rxd_Data==54)                              //if received
data is '6' then start profiling for DMA memset function
           {
                Rxd_Data=0;
                dma_memset_test(10);
                LOG(INFO,"dma-memset for 10 bytes");//log INFO to
terminal
           }

           if(Rxd_Data==48)                              //if received
data is '0' then stop profiling
           {
                Rxd_Data=0;
                LOG(PROFILING_COMPLETED,NULL);        //log INFO to
terminal
                break;
           }
     }
}
```

3) project3.h

```
/*****************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file project3.h
 * @brief This file includes logging functions
```

```
 * @date December 3, 2017
 *
 * long description - The logger.c file includes functions to -
 *                      1) project3() to initialise logging functions
 *                      2) data() to log data receive and data
analysis
 *                      3) profiling() to perform profiling function
on library, NON-DMA and DMA
 *                         memmove and memmset functions
 *

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

#ifndef SOURCES_PROJECT3_H_
#define SOURCES_PROJECT3_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include "rtc.h"
#include "logger.h"
#include "logbuf.h"
#include "uart.h"
#include "circbuf.h"
#include "gpio.h"
#include "dma.h"
#include "profiler.h"
#include "test.h"

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

#define NON_BLOCKING_MODE             (0x43)
#define BLOCKING_MODE                 (0x44)

/*------------------------extern variables---------------------------
*/
extern uint8_t Tx_Data;
extern uint8_t Rx_Data;
extern uint8_t Rxd_Data;
extern uint8_t character_count[4];
extern uint8_t char_str[4][10];
extern uint8_t *num;
extern uint8_t *alpha;
extern uint8_t *punc;
extern uint8_t *misc;
extern uint8_t time[4];
extern uint8_t t;
```

```
/********************************** project3()
*********************************************************
 *
 * @name   -  project3()
 * @brief  -  function to pass logs to log buffer and then print it to
terminal through terminal
 *
 * long description - This function takes data input from user and
depending on the received data
 *                                it performs various logging functions
such as data analysis and profiling
 *
 * @return -  void
 *

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

void project3();

/*********************************** data()
*********************************************************
 *
 * @name   -  data()
 * @brief  -  function to perform data analysis on the data received
through uart
 *
 * @return -  void
 *

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

void data();

/*********************************** profiling()
*********************************************************
 *
 * @name   -  profiling()
 * @brief  -  function to perform profiling operation on library, NON-
DMA and DMA memmove and memset functions
 *
 * @return -  void
 *

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

void profiling();

#endif /* SOURCES_PROJECT3_H_ */
```

4) project3_1.c

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file project3_1.c
 * @brief This file includes logging functions for host and beaglebone
black
 * @date November 23, 2017
 *
 * long description - The logger.c file includes functions to -
 *                      1) project3_1()
 *                      2) data()
 *                      3) profiling()
 *

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

#include "project3_1.h"

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

uint8_t character_count[4]={0,0,0,0};
uint8_t char_str[4][10];
char value;
uint32_t t = 1000000;
time_t t0;

/*********************************
project3_1()*************************************************************
 *
 * @name   -  project3_1()
 * @brief  -  function to initialize log and other logging functions
 *
 * long description - This function executes logging for host and
beaglebone black
 *
 * @return -  void
 *

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

/********************************** project3_1() function definition
*********************************/

void project3_1(){
    uint8_t i;
    uint8_t j;
```

```
      while(1)
      {
          printf("\n  %s  \t",ctime(&t0));
          printf("Initialize logger (1/0):  ");
          scanf("%d",&i);
            //read value from user
          if(i==1){
                      //if input is 1 then initialize log
             printf("\n  %s  \t",ctime(&t0));
             printf(" Log Initialized         %d\n\n",LOG_INITIALIZED);
      //print log initialized
             printf("  %s  \t",ctime(&t0));
             printf(" System Initialized
%d\n\n",SYSTEM_INITIALIZED);     //print system initialized

             while(1){
                 scanf("%d",&j);

                 if(j==1){
                 //if j==1 enable data input
                     j=0;
                     printf("  %s  \t",ctime(&t0));
                     printf(" Data Input Enabled
%d\n\n",DATA_INPUT_ENABLED);     //print data input enabled
                     data();
                 }
                 if(j==2){
                 //if j==2 enable profiling
                     j=0;
                     printf("  %s  \t",ctime(&t0));
                     printf(" Profiling Started
%d\n\n",PROFILING_STARTED);//print profiling started
                     profiling();
                     printf("  %s  \t",ctime(&t0));
                     printf(" Profiling Completed
%d\n\n",PROFILING_COMPLETED);    //print profiling completed
                 }
                 if(j==3){
                 //if j==3 exit loop
                     j=0;
                     break;
                 }
              }
             printf("  %s  \t",ctime(&t0));
             printf(" System Halted           %d\n\n",SYSTEM_HALTED);
      //print system halted
       }
          if(i==0){
            break;
       }
      }
}
```

```
/*********************************
data()************************************************************
 *
 * @name    -  data()
 * @brief   -  function to take data input and display data analysis
 *
 * long description - This function takes data input from user and
performs data analysis
 *
 * @return -  void
 *

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

/********************************* data() function definition
*********************************/

void data(){
    while(1){
        scanf("\n%c",&value);

        //----- Check for Numeric characters  ------------------------
-------------------- ------
        if((value>47) && (value<58)){
                            //check ascii value for numeric character
            value=atoi(&value);
            printf("  %s  \t",ctime(&t0));
            printf(" Data Received           %d
%d\n\n",DATA_RECEIVED,value);    //print the received character
                character_count[0]++;
                value=0;
        }
        //----- Check for Alphabetic characters  ---------------------
-------------------- ------
        else if(((value>64) && (value<91)) || ((value>96) &&
(value<123))){             //check ascii value for alphabetic
character
            printf("  %s  \t",ctime(&t0));
            printf(" Data Received           %d
%c\n\n",DATA_RECEIVED,value);    //print the received character
                character_count[1]++;
                value=0;
        }
        //----- Check for Punctuation characters  --------------------
--------------------- ----
        else if(((value>32) && (value<46))||((value>57) &&
(value<61))||((value>90) && (value<97))||((value>122) &&
(value<127))){        //check ascii value for punctuation character
                printf("  %s  \t",ctime(&t0));
```

```
                printf(" Data Received          %d
%c\n\n",DATA_RECEIVED,value);    //print the received character
                character_count[2]++;
                value=0;
        }
        //----- Check for Miscellaneous characters  ------------------
--------------------------
        else if(((value>90)&&(value<97))||(value==127)||(value==27)){
                //check ascii value for miscellaneous characters
                printf("  %s  \t",ctime(&t0));
                printf(" Data Received          %d
%c\n\n",DATA_RECEIVED,value);    //print the received character
                character_count[3]++;
             value=0;
        }
        //----- Check for . character - ------------------------------
------------------ ------
        else if(value==46){
                                //if '.' is pressed print analysis
                printf("  %s  \t",ctime(&t0));
                printf(" Data Analysis Started
%d\n\n",DATA_ANALYSIS_STARTED);       //print analysis started
                printf("  %s  \t",ctime(&t0));
                printf(" Data Alpha Count        %d
%d\n\n",DATA_ALPHA_COUNT,character_count[1]);    //print alpha count
                printf("  %s  \t",ctime(&t0));
                printf(" Data Numeric Count      %d
%d\n\n",DATA_NUMERIC_COUNT,character_count[0]);  //print numeric count
                printf("  %s  \t",ctime(&t0));
                printf(" Data Punc Count         %d
%d\n\n",DATA_PUNCTUATION_COUNT,character_count[2]);    //print
punctuation count
                printf("  %s  \t",ctime(&t0));
                printf(" Data Misc Count         %d
%d\n\n",DATA_MISC_COUNT,character_count[3]);     //print miscellaneous
count
                printf("  %s  \t",ctime(&t0));
                printf(" Data Analysis Completed
%d\n\n",DATA_ANALYSIS_COMPLETED);     //print analysis completed
             value=0;
        }

        if(value==47){
                //if '/' is pressed disable input
                value=0;
                printf("  %s  \t",ctime(&t0));
                printf(" Data Input Disabled
%d\n\n",DATA_INPUT_DISABLED);    //print input disabled
                break;
        }
    }
}
```

```
/*********************************
profiling()*****************************************************************
 *
 * @name    -  data()
 * @brief   -  function to take data input and display data analysis
 *
 * long description - This function takes data input from user and
performs data analysis
 *
 * @return -  void
 *

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

/********************************* profiling() function definition
*********************************/

void profiling(){
        lib_memmove_test(10);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memmove for 10
bytes\n\n",INFO);     //print info
        lib_memmove_test(100);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memmove for 100
bytes\n\n",INFO);     //print info
        lib_memmove_test(1000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memmove for 1000
bytes\n\n",INFO);     //print info
        lib_memmove_test(5000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memmove for 5000
bytes\n\n",INFO);     //print info
        lib_memset_test(10);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memset for 10
bytes\n\n",INFO);          //print info
        lib_memset_test(100);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memset for 100
bytes\n\n",INFO);     //print info
        lib_memset_test(1000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memset for 1000
bytes\n\n",INFO);     //print info
        lib_memset_test(5000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     lib-memset for 5000
bytes\n\n",INFO);     //print info
```

```
        my_memmove_test(10);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memmove for 10
bytes\n\n",INFO);          //print info
        my_memmove_test(100);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memmove for 100
bytes\n\n",INFO);     //print info
        my_memmove_test(1000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memmove for 1000
bytes\n\n",INFO);     //print info
        my_memmove_test(5000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memmove for 5000
bytes\n\n",INFO);     //print info
        my_memset_test(10);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memset for 10
bytes\n\n",INFO);          //print info
        my_memset_test(100);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memset for 100
bytes\n\n",INFO);          //print info
        my_memset_test(1000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memset for 1000
bytes\n\n",INFO);     //print info
        my_memset_test(5000);
        printf("  %s  \t",ctime(&t0));
        printf(" Info                    %d     my-memset for 5000
bytes\n\n",INFO);     //print info
}

/*********************************
lib_memmove_test()*****************************************************
******
 *
 * @name   -  lib_memmove_test()
 * @brief  -  function to perform profiling on library memmove
function
 *
 *
 * @return -  void
 *

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

/********************************* lib_memmove_test() function
definition *********************************/
```

```
void lib_memmove_test(uint32_t len){
    uint32_t i=0;
      uint8_t source[len];                    //initialize source array
      while(i<(len))
      {
            source[i]=i;
            i++;
      }
      uint8_t destination[len];               //initialize destination
array

      clock_t begin = clock();
      memmove(source,destination,len); //library memmove function
      clock_t end = clock();
      double time1 = (double)((end - begin)*t) / CLOCKS_PER_SEC;
      //calculate time
    printf("  %s  \t",ctime(&t0));
      printf(" Profiling Result        %d
%f\n\n",PROFILING_RESULT,time1); //print profiling result
}

/*********************************
lib_memset_test()*****************************************************
*****
 *
 * @name   -  lib_memset_test()
 * @brief  -  function to perform profiling on library memset function
 *
 *
 * @return -  void
 *

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

/********************************** lib_memset_test() function
definition *********************************/

void lib_memset_test(uint32_t len){
    uint8_t val = 10;
      uint8_t destination[len];               //initialize destination
array

      clock_t begin = clock();
      memset(destination,val,len);          //library memset function
      clock_t end = clock();
      double time1 = (double)((end - begin)*t) / CLOCKS_PER_SEC;
      //calculate time
    printf("  %s  \t",ctime(&t0));
      printf(" Profiling Result        %d
%f\n\n",PROFILING_RESULT,time1); //print profiling result
}
```

```
/*********************************
my_memmove_test()*****************************************************
*****
 *
 * @name    -  my_memmove_test()
 * @brief   -  function to perform profiling on my_memmove function
 *
 *
 * @return -  void
 *

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

/********************************* my_memmove_test() function
definition *********************************/

void my_memmove_test(uint32_t len){
    uint32_t i=0;
      uint8_t source[len];                  //initialize source array
      while(i<(len))
      {
            source[i]=i;
            i++;
      }
      uint8_t destination[len];             //initialize destination
array

      clock_t begin = clock();
      my_memmove(source,destination,len);         //my_memmove fuction
      clock_t end = clock();
      double time1 = (double)((end - begin)*t) / CLOCKS_PER_SEC;
      //calculate time
    printf("  %s  \t",ctime(&t0));
      printf(" Profiling Result        %d
%f\n\n",PROFILING_RESULT,time1); //print profiling result
}

/*********************************
my_memset_test()******************************************************
****
 *
 * @name    -  my_memset_test()
 * @brief   -  function to perform profiling on my_memset function
 *
 *
 * @return -  void
 *

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

```
/********************************** my_memset_test() function
definition **********************************/

void my_memset_test(uint32_t len){
    uint32_t val =10;

     uint8_t destination[len];              //initialize destination
array

     clock_t begin = clock();
     my_memset(val,destination,len);             //my_memset fuction
     clock_t end = clock();
     double time1 = (double)((end - begin)*t) / CLOCKS_PER_SEC;
     //calculate time
    printf("  %s  \t",ctime(&t0));
     printf(" Profiling Result        %d
%f\n\n",PROFILING_RESULT,time1); //print profiling result
}
```

5) project3_1.h

```
/*****************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file project3_1.h
 * @brief This file includes logging functions for host and beaglebone
black
 * @date November 23, 2017
 *
 * long description - The logger.c file includes functions to -
 *                      1) project3_1()
 *                      2) data()
 *                      3) profiling()
 *

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


#ifndef SOURCES_PROJECT3_1_H_
#define SOURCES_PROJECT3_1_H_

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include "memory.h"
```

```
/*********************************enum for LOG
IDs************************************************/

typedef enum{
      LOG_INITIALIZED = 100,
      GPIO_INITIALIZED,
      SYSTEM_INITIALIZED,
      SYSTEM_HALTED,
      INFO,
      WARNING,
      ERROR,
      PROFILING_STARTED,
      PROFILING_RESULT,
      PROFILING_COMPLETED,
      DATA_RECEIVED,
      DATA_ANALYSIS_STARTED,
      DATA_ALPHA_COUNT,
      DATA_NUMERIC_COUNT,
      DATA_PUNCTUATION_COUNT,
      DATA_MISC_COUNT,
      DATA_ANALYSIS_COMPLETED,
      HEARTBEAT,
      DATA_INPUT_DISABLED,
      DATA_INPUT_ENABLED
}Log_Status;


/*********************************
project3_1()*************************************************************
 *
 * @name   -  project3_1()
 * @brief  -  function to initialize log and other logging functions
 *
 * long description - This function executes logging for host and
beaglebone black
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void project3_1();

/*********************************
data()*******************************************************
 *
 * @name   -  data()
 * @brief  -  function to take data input and display data analysis
 *
 * long description - This function takes data input from user and
performs data analysis
 *
```

```
 * @return -  void
 *

*****************************************************************************
*******************************/
void data();

/*********************************
profiling()*****************************************************************
 *
 * @name   -  data()
 * @brief  -  function to take data input and display data analysis
 *
 * long description - This function takes data input from user and
performs data analysis
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void profiling();


/*********************************
lib_memmove_test()**************************************************************
******
 *
 * @name   -  lib_memmove_test()
 * @brief  -  function to perform profiling on library memmove
function
 *
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void lib_memmove_test(uint32_t len);

/*********************************
lib_memset_test()***************************************************************
*****
 *
 * @name   -  lib_memset_test()
 * @brief  -  function to perform profiling on library memset function
 *
 *
 * @return -  void
 *
```

```
*****************************************************************
*******************************/
void lib_memset_test(uint32_t len);

/*********************************
my_memmove_test()*****************************************************
*****
 *
 * @name    -  my_memmove_test()
 * @brief   -  function to perform profiling on my_memmove function
 *
 *
 * @return -  void
 *

*****************************************************************
*******************************/
void my_memmove_test(uint32_t len);

/*********************************
my_memset_test()******************************************************
****
 *
 * @name    -  my_memset_test()
 * @brief   -  function to perform profiling on my_memset function
 *
 *
 * @return -  void
 *

*****************************************************************
*******************************/
void my_memset_test(uint32_t len);

#endif /* SOURCES_PROJECT3_1_H_ */
```

6) logger.c

```
/*****************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file logger.c
 * @brief This file includes logging functions
 * @date November 23, 2017
 *
 * long description - The logger.c file includes functions to -
 *                      1) pass log structure to log
buffer(log_pass())
 *                      2) determine log id and pass appropriate
payload (log_id())
```

```
 *                      3) log string of characters (log_item())
 *                      4) remove (flush) log buffer data and send it
by UART channel (log_flush())
 *                      5) log string of characters (log_string())
 *                      6) log a length of data bytes (log_data())
 *                      7) log an integer (log_int())
 *                      8) send string of data through UART channel
(uart_flush())
 *
 *

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

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


Log_t log_buffer;
uint8_t Log_Tx_Data;

uint8_t log_tag[20][26] =          {"<LOG INITIALIZED>     ",
                                                "<GPIO INITIALIZED>
     ",
                                                "<SYSTEM INITIALIZED>
     ",
                                                "<SYSTEM HALTED>
     ",
                                                "<INFO>
     ",
                                                "<WARNING>
     ",
                                                "<ERROR>
     ",
                                                "<PROFILING STARTED>
     ",
                                                "<PROFILING RESULT>
     ",
                                                "<PROFILING
COMPLETED> ",
                                                "<DATA RECIEVED>
     ",
                                                "<DATA ANALYSIS
STARTED>   ",
                                                "<DATA ALPHA COUNT>
",
                                                "<DATA NUMERIC COUNT>
     ",
                                                "<DATA PUNCTUATION
COUNT>",
```

```
                                                    "<DATA MISC COUNT>
      ",
                                                    "<DATA ANALYSIS
COMPLETED>",
                                                    "<HEARTBEAT>
      ",
                                                    "<DATA INPUT
DISABLED>  ",
                                                    "<DATA INPUT ENABLED>
      "};

/*********************************
log_pass()*************************************************************
 *
 * @name   -  log_pass()
 * @brief  -  function to passes log structure to log buffer
 * @param  -  id0 : enum for loggger status id
 * @param  -  *log_payload: pointer to payload
 * @param  -  length: length of payload
 *
 * long description - This function passes log struct to log buffer.
 *
 * @return -  void
 *

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

/********************************** log_pass() function definition
*********************************/

void log_pass(Log_Status id0,uint8_t *log_payload, uint8_t length){

      //START_CRITICAL();
      Log_Struct log_struct;                              // log structure
declared
      log_struct.log_id0 = id0;                     // log id
      log_struct.timestamp = RTC->TSR;       // getting timestamp by
reading TSR register
      log_struct.payload = log_payload;           // passing log
payload ptr
      log_struct.len_payload = length;       // passing length of
payload

      log_item(&log_struct);                              // calling
log_item to add log structure to log buffer
      //END_CRITICAL();
      //log_flag = 0x78;
}

/*********************************
log_flush()************************************************************
```

```
 *
 * @name   -  log_flush()
 * @brief  -  function to remove (flush) log buffer data and send it
by UART channel.
 * @param  -  void
 *
 * long description - This function removes (flush) log buffer data
and send it by UART channel
 *                    in blocking mode.
 *
 * @return -  void
 *

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

/*********************************** log_flush() function definition
*********************************/

void log_flush(void){
     while(Log_buffer_is_empty(&log_buffer) != LOG_BUFFER_EMPTY){
//wait until log buffer is empty
          Log_buffer_remove_item(&log_buffer, &Log_Tx_Data);
// write data to log_tx_data variable
          UART_send(&Log_Tx_Data);
       // Send log_tx_data through UART channel
     }
}

/************************************
log_id()*************************************************************
 *
 * @name   -  log_id()
 * @brief  -  function to determine log id and pass appropriate
payload
 * @param  -  id: enum for loggger status id
 * @param  -  str: payload pointer
 *
 * long description - This function determine log id and pass
appropriate payload.
 *
 * @return -  void
 *

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

/************************************ log_id() function definition
*********************************/


void log_id(Log_Status id,uint8_t *str){
```

```
switch(id){
     case LOG_INITIALIZED:
//matching log id to pass the required payload
          log_pass(LOG_INITIALIZED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
         break;

     case GPIO_INITIALIZED:
          log_pass(GPIO_INITIALIZED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
    break;

     case SYSTEM_INITIALIZED:
          log_pass(SYSTEM_INITIALIZED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
       break;

     case SYSTEM_HALTED:
          log_pass(SYSTEM_HALTED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
```

```
            break;

      case INFO:
            log_pass(INFO,NULL,0);
            if(log_mode != 0x43){
// send data in blocking mode
                  log_flush();}
//send data using log flush if interrupt mode is not enabled
            else{
                  UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                  UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
            delay();
            log_string(str);
            break;

      case WARNING:
            log_pass(WARNING,NULL,0);
            if(log_mode != 0x43){
// send data in blocking mode
                  log_flush();}
//send data using log flush if interrupt mode is not enabled
            else{
                  UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                  UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
            delay();
            log_string(str);
            break;

      case ERROR:
            log_pass(ERROR,NULL,0);
            if(log_mode != 0x43){
// send data in blocking mode
                  log_flush();}
//send data using log flush if interrupt mode is not enabled
            else{
                  UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                  UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
            delay();
            log_string(str);
            break;

      case PROFILING_STARTED:
            log_pass(PROFILING_STARTED,NULL,0);
            if(log_mode != 0x43){
// send data in blocking mode
```

```
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case PROFILING_RESULT:
           log_pass(PROFILING_RESULT,&time[1],3);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case PROFILING_COMPLETED:
           log_pass(PROFILING_COMPLETED,NULL,0);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_RECEIVED:
           log_pass(DATA_RECEIVED,&Rxd_Data,1);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_ANALYSIS_STARTED:
           log_pass(DATA_ANALYSIS_STARTED,NULL,0);
           if(log_mode != 0x43){
// send data in blocking mode
```

```
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_ALPHA_COUNT:
           log_pass(DATA_ALPHA_COUNT,alpha,character_count[1]);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_NUMERIC_COUNT:
           log_pass(DATA_NUMERIC_COUNT,num,character_count[0]);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_PUNCTUATION_COUNT:

log_pass(DATA_PUNCTUATION_COUNT,punc,character_count[2]);
           if(log_mode != 0x43){
// send data in blocking mode
                log_flush();}
//send data using log flush if interrupt mode is not enabled
           else{
                UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
                UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
           break;

      case DATA_MISC_COUNT:
           log_pass(DATA_MISC_COUNT,misc,character_count[3]);
```

```
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
          break;

     case DATA_ANALYSIS_COMPLETED:
          log_pass(DATA_ANALYSIS_COMPLETED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
          break;

     case HEARTBEAT:
          log_pass(HEARTBEAT,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
          break;

     case DATA_INPUT_DISABLED:
          log_pass(DATA_INPUT_DISABLED,NULL,0);
          if(log_mode != 0x43){
// send data in blocking mode
               log_flush();}
//send data using log flush if interrupt mode is not enabled
          else{
               UART0_C2 &= ~UART0_C2_TIE_MASK;
// disable  UART Tx interrupt
               UART0_C2 |= UART0_C2_TIE_MASK;}
// enable UART Tx interrupt
          break;

     case DATA_INPUT_ENABLED:
          log_pass(DATA_INPUT_ENABLED,NULL,0);
```

```
                if(log_mode != 0x43){
      // send data in blocking mode
                      log_flush();}
      //send data using log flush if interrupt mode is not enabled
                else{
                      UART0_C2 &= ~UART0_C2_TIE_MASK;
      // disable  UART Tx interrupt
                      UART0_C2 |= UART0_C2_TIE_MASK;}
      // enable UART Tx interrupt
                break;

           default:
             break;
      }
}

/*********************************
log_item()*************************************************************
 *
 * @name   -  log_item()
 * @brief  -  function to add log struct to log buffer
 * @param  -  *ab: pointer to log structure
 *
 * long description - This function adds log struct to log buffer.
 *
 * @return -  void
 *

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

/*********************************** log_item() function definition
*********************************/

void log_item(Log_Struct *ab){

      uint8_t buffer[80];
      //buffer to store ascii values and add them to log buffer
      uint8_t i;
      for(i=0;i<10;i++)
      {
            buffer[i]=0;
      }
      uint8_t checksum;
      // checksum variable
      my_itoa(ab->timestamp,buffer,10);
      // converts timestamp to ascii string and store in buffer
      i = 1;
                while(buffer[i] != 0){
      //adding timestamp to log buffer
                      Log_buffer_add_item(&log_buffer, &buffer[i]);
```

```
                checksum ^= buffer[i];
    // calculating checksum
                i++;
                buffer[i-1] = 0;
           }
    for(i=0;i<10;i++)
    {
         buffer[i]=0;
    }
    i=' ';
    Log_buffer_add_item(&log_buffer,&i);
    uint8_t y;
    y = 0;
    y = (uint8_t)(ab->log_id0);
    my_itoa(y,buffer,10);

    i = '\t';
         Log_buffer_add_item(&log_buffer,&i);

    i = 0;
         while(log_tag[y-100][i] != 0){
    // adding log tag to log buffer
           checksum ^= log_tag[y-100][i];
           Log_buffer_add_item(&log_buffer, &log_tag[y-100][i]);
           i++;
         }
    i = '\t';
    Log_buffer_add_item(&log_buffer,&i);


    i = 1;
         while(buffer[i] != 0){
           checksum ^= buffer[i];
           Log_buffer_add_item(&log_buffer, &buffer[i]);
    //adding log id to log buffer
           i++;
           buffer[i-1] = 0;
         }
         for(i=0;i<10;i++)
         {
           buffer[i]=0;
         }
    //Log_buffer_add_item(&log_buffer, &bu);

    i='   ';
    Log_buffer_add_item(&log_buffer,&i);

    if(ab->payload != NULL){
    //pass payload and length only if payload is not null
         if(ab->log_id0==PROFILING_RESULT || ab-
>log_id0==DATA_RECEIVED)
         {
```

```
      //my_itoa(ab->len_payload,buffer,10);
      //i=1;
      while(buffer[i] != 0){
      //Log_buffer_add_item(&log_buffer, &buffer[i]);
      i++;
      //buffer[i-1] = 0;
      }
      i = ' ';
      Log_buffer_add_item(&log_buffer,&i);
      Log_buffer_add_item(&log_buffer,&i);
      }
      else
      {
            my_itoa(ab->len_payload,buffer,10);
            i=1;
            while(buffer[i] != 0){
                  checksum ^= buffer[i];
            Log_buffer_add_item(&log_buffer, &buffer[i]);
            i++;
            //buffer[i-1] = 0;
      }
            for(i=0;i<10;i++)
            {
                  buffer[i]=0;
            }
            i = ' ';
            Log_buffer_add_item(&log_buffer,&i);
            Log_buffer_add_item(&log_buffer,&i);
      }
      for(i=0;i<ab->len_payload;i++){
            checksum ^= *(ab->payload+i);
            Log_buffer_add_item(&log_buffer,ab->payload +i);
      }
}

else{
      i = ' ';
      Log_buffer_add_item(&log_buffer,&i);
      Log_buffer_add_item(&log_buffer,&i);
}
i = ' ';
Log_buffer_add_item(&log_buffer,&i);
Log_buffer_add_item(&log_buffer,&i);
my_itoa(checksum,buffer,10);
Log_buffer_add_item(&log_buffer,&buffer[1]);
for(i=0;i<10;i++)
{
      buffer[i]=0;
}
i = '   ';
Log_buffer_add_item(&log_buffer,&i);
```

```
      if(ab->log_id0!=INFO){
            i='\r';
            Log_buffer_add_item(&log_buffer,&i);
            i='\n';
            Log_buffer_add_item(&log_buffer,&i);
            i='\r';
            Log_buffer_add_item(&log_buffer,&i);
            i='\n';
            Log_buffer_add_item(&log_buffer,&i);
            }
}

/*********************************
log_string()*************************************************************
 *
 * @name   -  log_string()
 * @brief  -  function to log string of characters
 * @param  -  *str: pointer to string
 *
 * long description - This function logs string of characters
 *
 * @return -  void
 *

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

/*********************************** log_string() function
definition *********************************/

void log_string(uint8_t *str){
      uint8_t d;
      uint8_t i;
      while(*str != 0){                     // condition to send string
data through UART channel
            d = *str;
            UART_send(&d);                  // send string through UART
            str++;
      }
      i='\r';
      UART_send(&i);
      i='\n';
      UART_send(&i);
      i='\r';
      UART_send(&i);
      i='\n';
      UART_send(&i);
}

/*********************************
log_data()***************************************************************
 *
```

```
 * @name   -  log_data()
 * @brief  -  function to log a length of data bytes
 * @param  -  *data: pointer to log structure
 * @param  -  length: length of bytes
 *
 * long description - This function logs a length of data bytes.
 *
 * @return -  void
 *

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

/*********************************** log_data() function definition
*********************************/

void log_data(uint8_t *data, uint8_t length){
      uint8_t i = '\t';
      UART_send(&i);
      i = 0;
      uint8_t l;
      while(i<length){                              // condition to send
length of data bytes through UART channel
            l = *(data + i);
            UART_send(&l);
            i++;
      }
}

/*********************************
log_int()*************************************************************
 *
 * @name   -  log_int()
 * @brief  -  function to log an integer
 * @param  -  num: integer to be logged
 *
 * long description - This function logs integer.
 *
 * @return -  void
 *

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

/*********************************** log_int() function definition
*********************************/

void log_int(int32_t num){
      uint8_t i = '\t';
      UART_send(&i);
      uint8_t buffer[10];
      uint8_t o = 1;
```

```
	my_itoa(num,buffer,10);
	while(buffer[o] != 0){                                          //
condition to send integer ascii string through UART channel
		UART_send(&buffer[o]);
		o++;
	}
}

/*********************************
uart_flush()************************************************************
 *
 * @name   -  uart_flush()
 * @brief  -  function to send string of data through UART channel
 * @param  -  *str: pointer to string of data
 *
 * long description - This function sends string of data through UART
channel
 *
 * @return -  void
 *

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

/*********************************** uart_flush() function
definition ***********************************/

void uart_flush(uint8_t *str){
	uint8_t d;
	uint8_t i = '\t';
	UART_send(&i);
	while(*str != 0){                       // send string through UART
using polling
		d = *str;
		UART_send(&d);
		str++;
	}
}
```

7) logger.h

```
/*****************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file logger.h
 * @brief This file includes function declarations for logging
functions
 * @date November 23, 2017
 *
 * long description - The logger.h file includes functions to -
```

```
 *                      1) pass log structure to log
buffer(log_pass())
 *                      2) determine log id and pass appropriate
payload (log_id())
 *                      3) log string of characters (log_item())
 *                      4) remove (flush) log buffer data and send it
by UART channel (log_flush())
 *                      5) log string of characters (log_string())
 *                      6) log a length of data bytes (log_data())
 *                      7) log an integer (log_int())
 *                      8) send string of data through UART channel
(uart_flush())
 *
 *

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

#ifndef SOURCES_LOGGER_H_
#define SOURCES_LOGGER_H_

#include <stdint.h>
#include "MKL25Z4.h"
#include <stdlib.h>
#include "logbuf.h"
#include "uart.h"
#include "conversion.h"
#include "project3.h"
#include "spi.h"


#define LOG(id,str)                                          (log_id(id,str))
#define LOG_DATA(data,length)                 (log_data(*data,length))
#define LOG_STRING(str)                            (log_string(*str))
#define LOG_INT(num)                              (log_int(num))
#define LOG_FLUSH()                                (log_flush())

extern Log_t log_buffer;
extern uint8_t Log_Tx_Data;


/*****************************enum for LOG
IDs************************/

typedef enum{
      LOG_INITIALIZED = 100,
      GPIO_INITIALIZED,
      SYSTEM_INITIALIZED,
      SYSTEM_HALTED,
      INFO,
      WARNING,
      ERROR,
```

```
      PROFILING_STARTED,
      PROFILING_RESULT,
      PROFILING_COMPLETED,
      DATA_RECEIVED,
      DATA_ANALYSIS_STARTED,
      DATA_ALPHA_COUNT,
      DATA_NUMERIC_COUNT,
      DATA_PUNCTUATION_COUNT,
      DATA_MISC_COUNT,
      DATA_ANALYSIS_COMPLETED,
      HEARTBEAT,
      DATA_INPUT_DISABLED,
      DATA_INPUT_ENABLED
}Log_Status;


/**********************Structure for
Logger*****************************/

typedef struct{

      Log_Status log_id0;
      uint32_t timestamp;
      uint16_t len_payload;
      uint8_t *payload;
      uint16_t checksum;

}Log_Struct;

/*********************************
log_pass()*************************************************************
 *
 * @name   -  log_pass()
 * @brief  -  function to passes log structure to log buffer
 * @param  -  id0 : enum for loggger status id
 * @param  -  *log_payload: pointer to payload
 * @param  -  length: length of payload
 *
 * long description - This function passes log struct to log buffer.
 *
 * @return -  void
 *

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

/********************************** log_pass() function declaration
*********************************/
void log_pass(Log_Status id0,uint8_t *log_payload, uint8_t length);

/*************************************
log_id()***************************************************************
```

```
 *
 * @name   -  log_id()
 * @brief  -  function to determine log id and pass appropriate
payload
 * @param  -  id: enum for loggger status id
 * @param  -  str: payload pointer
 *
 * long description - This function determine log id and pass
appropriate payload.
 *
 * @return -  void
 *

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

/*********************************** log_id() function declaration
***********************************/

void log_id(Log_Status id,uint8_t *str);

/*********************************
log_item()*************************************************************
 *
 * @name   -  log_item()
 * @brief  -  function to add log struct to log buffer
 * @param  -  *ab: pointer to log structure
 *
 * long description - This function adds log struct to log buffer.
 *
 * @return -  void
 *

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

/*********************************** log_item() function declaration
***********************************/

void log_item(Log_Struct *ab);

/*********************************
log_flush()*************************************************************
 *
 * @name   -  log_flush()
 * @brief  -  function to remove (flush) log buffer data and send it
by UART channel.
 * @param  -  void
 *
 * long description - This function removes (flush) log buffer data
and send it by UART channel
 *                    in blocking mode.
```

```
 *
 * @return -  void
 *

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

/*********************************** log_flush() function declaration
**********************************/


void log_flush(void);

/*********************************
log_string()*************************************************************
 *
 * @name   -  log_string()
 * @brief  -  function to log string of characters
 * @param  -  *str: pointer to string
 *
 * long description - This function logs string of characters
 *
 * @return -  void
 *

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

/************************************ log_string() function
declaration **********************************/

void log_string(uint8_t *str);

/*********************************
log_data()***************************************************************
 *
 * @name   -  log_data()
 * @brief  -  function to log a length of data bytes
 * @param  -  *data: pointer to log structure
 * @param  -  length: length of bytes
 *
 * long description - This function logs a length of data bytes.
 *
 * @return -  void
 *

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

/************************************ log_data() function declaration
**********************************/
```

```
void log_data(uint8_t *data, uint8_t length);

/*********************************
log_int()*************************************************************
 *
 * @name   -  log_int()
 * @brief  -  function to log an integer
 * @param  -  num: integer to be logged
 *
 * long description - This function logs integer.
 *
 * @return -  void
 *

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

/*********************************** log_int() function declaration
**********************************/

void log_int(int32_t num);

/*********************************
uart_flush()**********************************************************
 *
 * @name   -  uart_flush()
 * @brief  -  function to send string of data through UART channel
 * @param  -  *str: pointer to string of data
 *
 * long description - This function sends string of data through UART
channel
 *
 * @return -  void
 *

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

/*********************************** uart_flush() function
declaration **********************************/

void uart_flush(uint8_t *str);



#endif /* SOURCES_LOGGER_H_ */
```

8) logbuf.c

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

```
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @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 "logbuf.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 ***********************************/

Log_Buffer_Status Log_buffer_add_item(Log_t *log, volatile uint8_t
*data)
```

```
{
    if(log->count==log->length)
    {
        return LOG_BUFFER_FULL;
    }
    else if(log->head==log->buffer_end && log->count<log->length)
    {
        log->head=log->buffer;
        *(log->head)=*data;
        log->count++;
        log->head++;
        return LOG_SUCCESS;
    }
    else
    {
        *(log->head)=*data;
        log->count++;
        log->head++;
        return LOG_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 *********************************/

Log_Buffer_Status Log_buffer_remove_item(Log_t *log, volatile uint8_t
*data)
{
    if(log->count==0)
    {
        return LOG_BUFFER_EMPTY;
    }
    else if(log->head==log->buffer && log->tail==log->buffer)
    {
```

```
        *data=*(log->tail);
        log->count--;
        return LOG_SUCCESS;
    }

    else if(log->tail==log->buffer_end)
    {
        *data=*(log->tail);
        log->tail=log->buffer;
        log->count--;
        return LOG_SUCCESS;
    }
    else
    {
        *data=*(log->tail);
        log->tail++;
        log->count--;
        return LOG_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
*********************************/

//Log_Buffer_Status Log_buffer_is_full(Log_t *log)


/********************************* 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
*********************************/
//Log_Buffer_Status Log_buffer_is_empty(Log_t *log)



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

Log_Buffer_Status Log_buffer_peek(Log_t *log, uint8_t peek_pos,
uint8_t *peek_ptr)
{
    if(((log->count) != 0) && (peek_pos <= (log->length)))
    {
        if(((log->head)+(peek_pos))<(log->buffer_end))
        {
            *peek_ptr= *((log->head)+(peek_pos));
        }
```

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

        return LOG_SUCCESS;
    }
    else if((log->count) == 0)
    {
        return LOG_BUFFER_EMPTY;
    }
    else
    {
        return LOG_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
*********************************/

Log_Buffer_Status Log_buffer_init(Log_t *log, uint16_t length)
{
    log->buffer = (uint8_t*)malloc(sizeof(uint8_t)*length);
    log->buffer_end = log->buffer + (sizeof(uint8_t)*length);
    log->head = log->buffer;
    log->tail = log->buffer;
    log->count = 0;
    log->length = length;
    return LOG_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
*********************************/

Log_Buffer_Status Log_buffer_destroy(Log_t *log)
{
    free(log->buffer);
    return LOG_SUCCESS;
}
```

9) logbuf.h

```
/*********************************************************************
*****************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @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 SOURCES_LOGBUF_H_
#define SOURCES_LOGBUF_H_

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

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include "MKL25Z4.h"

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

typedef struct{
uint8_t *buffer;
uint8_t *buffer_end;
uint8_t *head;
uint8_t *tail;
uint16_t length;
uint8_t count;
}Log_t;

/*----- Enum for Circular Buffer and UART ---------------------------
-----*/


typedef enum{
LOG_BUFFER_FULL=11,
LOG_BUFFER_EMPTY=80,
LOG_SUCCESS=13,
LOG_NULL_ERROR=0,
LOG_PEEK_LENGTH_ERROR=15,
LOG_TX_SUCCESS=16,
LOG_RX_SUCCESS=17,
LOG_TX_IRQ=18,
LOG_RX_IRQ=19
}Log_Buffer_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 *********************************/

Log_Buffer_Status Log_buffer_add_item(Log_t *log, volatile 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 *********************************/

Log_Buffer_Status Log_buffer_remove_item(Log_t *log, volatile 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
*********************************/

//Log_Buffer_Status Log_buffer_is_full(Log_t *log);
__attribute__((always_inline)) static inline Log_Buffer_Status
Log_buffer_is_full(Log_t *log)
{
    if (log->count == log->length)
        return LOG_BUFFER_FULL;
    else
        return LOG_NULL_ERROR;
}

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

//Log_Buffer_Status Log_buffer_is_empty(Log_t *log);

__attribute__((always_inline)) static inline Log_Buffer_Status
Log_buffer_is_empty(Log_t *log)
{
    if (log->count == 0)
        return LOG_BUFFER_EMPTY;
    else
        return LOG_NULL_ERROR;
}

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

Log_Buffer_Status Log_buffer_peek(Log_t *log, 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
*********************************/

Log_Buffer_Status Log_buffer_init(Log_t *log, uint16_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
*********************************/
Log_Buffer_Status Log_buffer_destroy(Log_t *log);



#endif /* SOURCES_LOGBUF_H_ */
```

10) dma.c

```
/*************************************************************************
*****************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file dma.c
 * @brief This source file includes UART function definitions
 * @date December 03, 2017
 *
 * long description - The dma.c file includes functions to -
 *                       1) move data from source address to
destination address using DMA (memmove_dma())
 *                       2) set a length of data bytes to a particular
value using DMA (memset_dma())
 *                       3) handle DMA interrupts (DMA0_IRQHandler())
 *
 *

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

#include "dma.h"

/********************************** memmove_dma()
*****************************************
 *
 * @name  memmove_dma
 * @brief function to move length of bytes data from source to
destination using DMA
 * @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.
 *        4) size - byte-transfer size
 *
 * 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) using DMA.
 *

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


 /*************************** memmove_dma() function definition
*********************************/

void memmove_dma(void *src, void *dst, uint32_t length, uint8_t size)
{
     SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;      //enable clock for DMA
     SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;         //enable clock for
DMAMUX

     DMAMUX0_CHCFG0 = 0x00;                              //disable DMA
MUX channel
     DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK;//clear DMA0 status bits
and DMA0 interrupt bit

     DMA_SAR0 = (uint32_t)src;                  //copy source address
     DMA_DAR0 = (uint32_t)dst;                  //copy destination
address

     DMA_DSR_BCR0 |= length;                    // number of bytes to
transfer
     DMA_DCR0 |= DMA_DCR_SINC_MASK;             // increment SAR
     DMA_DCR0 |= DMA_DCR_DINC_MASK;             // increment DAR

     if(size==EIGHT_BIT)
     {
          DMA_DCR0 |= EIGHT_BIT_SOURCE_MASK;                    //
source size: 8-bit, 16-bit or 32-bit
          DMA_DCR0 |= EIGHT_BIT_DESTINATION_MASK;               //
destination size: 8-bit, 16-bit or 32-bit
     }

     else if(size==SIXTEEN_BIT)
          {
               DMA_DCR0 |= SIXTEEN_BIT_SOURCE_MASK;        // source
size: 8-bit, 16-bit or 32-bit
               DMA_DCR0 |= SIXTEEN_BIT_DESTINATION_MASK;   //
destination size: 8-bit, 16-bit or 32-bit
```

```
            }

      else if(size==THIRTYTWO_BIT)
            {
                  DMA_DCR0 |= THIRTYTWO_BIT_SOURCE_MASK;      // source
size: 8-bit, 16-bit or 32-bit
                  DMA_DCR0 |= THIRTYTWO_BIT_DESTINATION_MASK; //
destination size: 8-bit, 16-bit or 32-bit
            }

      else
      {
            DMA_DCR0 |= EIGHT_BIT_SOURCE_MASK;                    //
source size: 8-bit, 16-bit or 32-bit
            DMA_DCR0 |= EIGHT_BIT_DESTINATION_MASK;               //
destination size: 8-bit, 16-bit or 32-bit
      }

      DMA_DCR0 |= DMA_DCR_AA_MASK;
      //enable auto align

      DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK;              //enable
DMA MUX channel

      DMA_DCR0 |= DMA_DCR_EINT_MASK;
      //enable DMA interrupt

      NVIC_SetPriority(DMA0_IRQn,0);
      NVIC_EnableIRQ(DMA0_IRQn);
      __enable_irq;

      DMA_DCR0 |= DMA_DCR_START_MASK;
      //start DMA transfer
}

/********************************** memset_dma()
***********************************************
 *
 * @name  memset_dma
 * @brief function to set a length of data bytes to a specified value
using DMA
 * @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
 *        4) size - byte-transfer size
 *
 * 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 using DMA.
 *
```

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

 /***************************** memset_dma() function definition
*********************************/

void memset_dma(uint8_t value, void *dst, uint32_t length, uint8_t
size)
{
      SIM_SCGC7 |= SIM_SCGC7_DMA_MASK;       //enable clock for DMA
      SIM_SCGC6 |= SIM_SCGC6_DMAMUX_MASK;         //enable clock for
DMAMUX

      DMAMUX0_CHCFG0 = 0x00;                              //disable DMA
MUX channel
      DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK;//clear DMA0 status bits
and DMA0 interrupt bit

      DMA_SAR0 = (uint32_t)&value;                //copy source address
      DMA_DAR0 = (uint32_t)dst;                   //copy destination
address

      DMA_DSR_BCR0 |= length;                     // number of bytes to
transfer
      DMA_DCR0 |= DMA_DCR_SINC_MASK;              // increment SAR
      DMA_DCR0 |= DMA_DCR_DINC_MASK;              // increment DAR

      if(size==EIGHT_BIT)
      {
            DMA_DCR0 |= EIGHT_BIT_SOURCE_MASK;               // source
size: 8-bit, 16-bit or 32-bit
            DMA_DCR0 |= EIGHT_BIT_DESTINATION_MASK;               //
destination size: 8-bit, 16-bit or 32-bit
      }

      else if(size==SIXTEEN_BIT)
            {
                  DMA_DCR0 |= SIXTEEN_BIT_SOURCE_MASK;  // source size:
8-bit, 16-bit or 32-bit
                  DMA_DCR0 |= SIXTEEN_BIT_DESTINATION_MASK;   //
destination size: 8-bit, 16-bit or 32-bit
            }

      else if(size==THIRTYTWO_BIT)
            {
                  DMA_DCR0 |= THIRTYTWO_BIT_SOURCE_MASK;// source size:
8-bit, 16-bit or 32-bit
                  DMA_DCR0 |= THIRTYTWO_BIT_DESTINATION_MASK; //
destination size: 8-bit, 16-bit or 32-bit
            }
```

```
      else
      {
            DMA_DCR0 |= EIGHT_BIT_SOURCE_MASK;                // source
size: 8-bit, 16-bit or 32-bit
            DMA_DCR0 |= EIGHT_BIT_DESTINATION_MASK;                //
destination size: 8-bit, 16-bit or 32-bit
      }

      DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_ENBL_MASK;   //enable DMA MUX
channel

      DMA_DCR0 |= DMA_DCR_EINT_MASK;              //enable DMA
interrupt

      NVIC_EnableIRQ(DMA0_IRQn);
      __enable_irq;

      DMA_DCR0 |= DMA_DCR_START_MASK;             //start DMA transfer

      //while(!(DMA_DSR_BCR0 & DMA_DSR_BCR_DONE_MASK));      //check if
transfer is completed
}

/********************************* DMA0_IRQHandler()
*********************************************************
 *
 * @name   -  DMA0_IRQHandler()
 * @brief  -  function to handle DMA interrupt
 * @param  -  none
 *
 * long description - This function handles DMA interrupt.
 *
 * @return -  void
 *

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

/******************************** DMA0_IRQHandler() function
definition ***********************************/

void DMA0_IRQHandler()
{
      DMA_DSR_BCR0 |= DMA_DSR_BCR_DONE_MASK;      //clear DMA0 status
bits and DMA0 interrupt bit
}
```

11) dma.h

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

```
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file dma.h
 * @brief This header file includes UART function declarations.
 * @date December 03, 2017
 *
 * long description - The dma.c file includes functions declarations
of functions to -
 *                      1) move data from source address to
destination address using DMA (memmove_dma())
 *                      2) set a length of data bytes to a particular
value using DMA (memset_dma())
 *                      3) handle DMA interrupts (DMA0_IRQHandler())
 *
 *

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

#ifndef SOURCES_DMA_H_
#define SOURCES_DMA_H_

#include "MKL25Z4.h"
#include <stdlib.h>
#include <stdint.h>
#include "project3.h"

#define EIGHT_BIT          8
#define SIXTEEN_BIT        16
#define THIRTYTWO_BIT 32

#define EIGHT_BIT_SOURCE_MASK          (DMA_DCR_SSIZE(1))
#define SIXTEEN_BIT_SOURCE_MASK        (DMA_DCR_SSIZE(2))
#define THIRTYTWO_BIT_SOURCE_MASK      (DMA_DCR_SSIZE(0))

#define EIGHT_BIT_DESTINATION_MASK          (DMA_DCR_DSIZE(1))
#define SIXTEEN_BIT_DESTINATION_MASK   (DMA_DCR_DSIZE(2))
#define THIRTYTWO_BIT_DESTINATION_MASK(DMA_DCR_DSIZE(0))

/******************************** memmove_dma()
*****************************************
 *
 * @name  memmove_dma
 * @brief function to move length of bytes data from source to
destination using DMA
 * @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.
 *        4) size - byte-transfer size
 *
 * 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) using DMA.
 *

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

 /***************************** memmove_dma() function declaration
*********************************/


void memmove_dma(void *src, void *dst, uint32_t length, uint8_t size);


/********************************* memset_dma()
***********************************************
 *
 * @name  memset_dma
 * @brief function to set a length of data bytes to a specified value
using DMA
 * @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
 *        4) size - byte-transfer size
 *
 * 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 using DMA.
 *

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

 /***************************** memset_dma() function declaration
*********************************/

void memset_dma(uint8_t value, void *dst, uint32_t length, uint8_t
size);


/********************************* DMA0_IRQHandler()
*****************************************************
 *
 * @name   -  DMA0_IRQHandler()
 * @brief  -  function to handle DMA interrupt
 * @param  -  none
 *
 * long description - This function handles DMA interrupt.
 *
```

```
 * @return -  void
 *
**************************************************************************
************************************/

/******************************** DMA0_IRQHandler() function
declaration *********************************/

void DMA0_IRQHandler();

#endif /* SOURCES_DMA_H_ */
```

12) circbuf.c

```
/**************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @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"

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

CB_t Tx_Buffer;
CB_t Rx_Buffer;
```

```
/********************************** 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, volatile uint8_t *data)
{
    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, volatile uint8_t *data)
{
    if(cb->count==0)
    {
        return BUFFER_EMPTY;
    }
    else if(cb->head==cb->buffer && cb->tail==cb->buffer)
    {
        *data=*(cb->tail);
        cb->count--;
        return SUCCESS;
    }

    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_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->length)))
    {
        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)
{
    cb->buffer = (uint8_t*)malloc(sizeof(uint8_t)*length);
    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)
{
    free(cb->buffer);
    return SUCCESS;
}
```

13) circbuf.h

```
/*****************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file circbuf.h
 * @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())
 *
 *

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

#ifndef SOURCES_CIRCBUF_H_
#define SOURCES_CIRCBUF_H_

#include <stdint.h>
#include "MKL25Z4.h"
#include <stdlib.h>
#include "logbuf.h"
#include "uart.h"
#include "conversion.h"
#include "logger.h"
#include "project3.h"

/*----- Structure for Circular Buffer --------------------------------
-----*/

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_EMPTY=2,
SUCCESS=3,
NULL_ERROR=0,
PEEK_LENGTH_ERROR=5

}CB_status;

extern CB_t Tx_Buffer;
extern CB_t Rx_Buffer;


/********************************* 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, volatile 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, volatile uint8_t *data);

/********************************** CB_is_full()
***************************************************
 *
 * @name   -  CB_is_full()
 * @brief  -  a static inline 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
***********************************/

__attribute__((always_inline)) static inline CB_status CB_is_full(CB_t
*cb)
{
    if (cb->count == cb->length)
        return BUFFER_FULL;
    else
        return NULL_ERROR;
}

/********************************** CB_is_empty()
***************************************************
 *
 * @name   -  CB_is_empty()
 * @brief  -  a static inline 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);
__attribute__((always_inline)) static inline CB_status
CB_is_empty(CB_t *cb)
{
    if (cb->count == 0)
        return BUFFER_EMPTY;
    else
        return NULL_ERROR;
}

/********************************** 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 /* SOURCES_CIRCBUF_H_ */
```

14) conversion.c

```
/*************************************************************************
*****************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @file conversion.c
 * @brief This file includes data conversion functions
 * @date October 2, 2017
 *
 * long decription - The conversion.c file includes data conversion
function for -
```

```
 *                      1) integer to ascii string (my_itoa())
 *
*****************************************************************************
*****************************/

/************* including header** libraries*************/

#include "conversion.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 definition
*************************************/
```

```
uint8_t 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;
    }
}
```

15) conversion.h

```
/*****************************************************************
*****************************
 *
 * @author Preshit Harlikar, Shivam Khandelwal
 * @file conversion.h
 * @brief This file includes data conversion functions
 * @date October 2, 2017
 *
 * long decription - The conversion.c file includes data conversion
function for -
 *                      1) integer to ascii string (my_itoa())
 *

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

```
#ifndef SOURCES_CONVERSION_H_
#define SOURCES_CONVERSION_H_

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

#include <stdio.h>
#include <stdint.h>
#include "project3.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.
/*****************************************************************
*******************************/

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

#endif /* SOURCES_CONVERSION_H_ */
```

16) gpio.c

```
/*****************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file gpio.c
 * @brief This file includes functions to initialize GPIO pin/port.
 * @date December 01, 2017
 *
 * long description - The gpio.c file includes functions to -
 *                      1) initialize GPIO for nrf
module(GPIO_nrf_init())
 *                      2) initialize GPIO for on-board
led(GPIO_led_init())
 *

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

#include "gpio.h"
#include "logger.h"

/******************************** GPIO_nrf_init()
*******************************************************
```

```
 *
 * @name   -  GPIO_nrf_init()
 * @brief  -  function to initialize GPIO for nrf module.
 * @param  -  none
 *
 * long description - This function initializes GPIO for nrf module
 *
 * @return -  void
 *

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

/********************************** GPIO_nrf_init() function
definition **********************************/

void GPIO_nrf_init()
{
     SIM->SCGC5 |= SIM_SCGC5_PORTD_MASK;         //enable clock for
PORT D

     PORTD_PCR0 |= PORT_PCR_MUX(1);                    //SPI0_PCS0
     PORTD_PCR1 |= PORT_PCR_MUX(2);                    //SPI0_SCK
     PORTD_PCR2 |= PORT_PCR_MUX(2);                    //SPI0_MOSI
     PORTD_PCR3 |= PORT_PCR_MUX(2);                    //SPI0_MISO
     PTD_BASE_PTR->PDDR |= 0x01;                       //Set pin 1 port
D i.e PCS as output direction
     LOG(GPIO_INITIALIZED,NULL);
}

/********************************** GPIO_led_init()
*************************************************************
 *
 * @name   -  GPIO_led_init()
 * @brief  -  function to initialize GPIO for on-board LED.
 * @param  -  none
 *
 * long description - This function initializes GPIO for on-board LED.
 *
 * @return -  void
 *

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

/********************************** GPIO_led_init() function
definition **********************************/

void GPIO_led_init()
{
     SIM_BASE_PTR->SCGC5 |= SIM_SCGC5_PORTB_MASK;// enable clock for
PORT B
```

```
     PORTB_BASE_PTR->PCR[18] = PORT_PCR_MUX(1);  // select pin 18 of
PORT B as RED LED
     PORTB_BASE_PTR->PCR[19] = PORT_PCR_MUX(1);  // select pin 18 of
PORT B as GREEN LED
     RED_LED_INIT;
     GREEN_LED_INIT;
}
```

17) gpio.h

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file gpio.h
 * @brief This file includes function declarations of functions to
initialize GPIO pin/port.
 * @date December 01, 2017
 *
 * long description - The gpio.h file includes functions declarations
of functions to -
 *                      1) initialize GPIO for nrf
module(GPIO_nrf_init())
 *                      2) initialize GPIO for on-board
led(GPIO_led_init())
 *

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

#ifndef SOURCES_GPIO_H_
#define SOURCES_GPIO_H_

#include "MKL25Z4.h"
#include "project3.h"

#define RED_LED_INIT       (GPIOB->PDDR |= 1 << 18)    //set port B pin
18 direction as output
#define GREEN_LED_INIT          (GPIOB->PDDR |= 1 << 19)    //set port
B pin 19 direction as output
#define BLUE_LED_INIT      (GPIOD->PDDR |= 1 << 1)          //set port
D pin 1 direction as output

#define RED_LED_ON              (GPIOB->PCOR |= 1 << 18)    //port B
pin 18 set as pin is active low
#define GREEN_LED_ON       (GPIOB->PCOR |= 1 << 19)    //port B pin 19
set as pin is active low
#define BLUE_LED_ON             (GPIOD->PCOR |= 1 << 1)
     //port D pin 1 set as pin is active low
```

```
#define RED_LED_OFF              (GPIOB->PSOR |= 1 << 18)    //port B
pin 18 clear as pin is active low
#define GREEN_LED_OFF      (GPIOB->PSOR |= 1 << 19)    //port B pin 19
clear as pin is active low
#define BLUE_LED_OFF       (GPIOD->PSOR |= 1 << 1)          //port D
pin 1 clear as pin is active low

#define RED_LED_TOGGLE           (GPIOB->PTOR |= 1 << 18)    //port B
pin 18 toggle
#define GREEN_LED_TOGGLE    (GPIOB->PTOR |= 1 << 19)    //port B pin 19
toggle
#define BLUE_LED_TOGGLE          (GPIOD->PTOR |= 1 << 1)
      //port D pin 1 toggle

/********************************** GPIO_nrf_init()
*********************************************************
 *
 * @name   -  GPIO_nrf_init()
 * @brief  -  function to initialize GPIO for nrf module.
 * @param  -  none
 *
 * long description - This function initializes GPIO for nrf module
 *
 * @return -  void
 *

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

/********************************** GPIO_nrf_init() function
declaration **********************************/

void GPIO_nrf_init();

/********************************** GPIO_led_init()
*********************************************************
 *
 * @name   -  GPIO_led_init()
 * @brief  -  function to initialize GPIO for on-board LED.
 * @param  -  none
 *
 * long description - This function initializes GPIO for on-board LED.
 *
 * @return -  void
 *

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

/********************************** GPIO_led_init() function
declaration **********************************/
```

```
void GPIO_led_init();



#endif /* SOURCES_GPIO_H_ */
```

18) 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) setting bytes of data to a specified
value(my_memset())

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

/************* including header libraries***************/

#include "memory.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.
 *

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


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


void my_memmove(uint8_t *src, uint8_t *dst, uint32_t length)
{
    uint32_t i;                                     // variable i
declared for loop condition

    if(dst<src || (src+length)<dst)                 // loop condition to
copy bytes
    {
      for(i=0;i<length;i++)
      {
            *(dst+i)=*(src+i);                      //Copying each byte
consecutively.
      }
    }

    else if(src<dst)                                // loop condition to
copy bytes
    {
      for(i=(length-1);i>=0;i--)
      {
            *(dst+i)=*(src+i);                      //Copying each byte
consecutively.
      }
    }
}

/********************************* 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.
 *
*****************************************************************
*************************/


void my_memset(uint32_t value, uint8_t *dst, uint32_t length)
{
      uint32_t i=0;
      for(i=0;i<length;i++)            // loop condition to set a
length of bytes to specified value.
      {
            *(dst+i)=value;                  // setting each byte
consecutively to the specified value.

      }
}



/******************************** my_memmove_optimized()
*****************************************
 *
 * @name  *my_memmove_optimized(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.
 *
*****************************************************************
*************************/
```

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


void my_memmove_optimized(uint8_t *src, uint8_t *dst, uint32_t length)
{
    uint32_t i;                                     // variable i
declared for loop condition

    if(dst<src || (src+length)<dst)                 // loop condition to
copy bytes
    {
      for(i=0;i<length;i++)
      {
            *(dst+i)=*(src+i);                      //Copying each byte
consecutively.
      }
    }

    else if(src<dst)                                // loop condition to
copy bytes
    {
      for(i=(length-1);i>=0;i--)
      {
            *(dst+i)=*(src+i);                      //Copying each byte
consecutively.
      }
    }
}

/********************************* my_memset_optimized()
*********************************************
 *
 * @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.
 *

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

```
void my_memset_optimized(uint32_t value, uint8_t *dst, uint32_t
length)
{
      uint32_t i=0;
      for(i=0;i<length;i++)            // loop condition to set a
length of bytes to specified value.
      {
            *(dst+i)=value;                  // setting each byte
consecutively to the specified value.

      }
}
```

19) memory.h

```
/**********************************************************************
*****************************
 *
 * @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) setting bytes of data to a specified
value(my_memset())

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

#ifndef SOURCES_MEMORY_H_
#define SOURCES_MEMORY_H_

/************* including header libraries***************/

#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include "project3.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.
 *
************************************************************************
*************************/

void my_memmove(uint8_t *src, uint8_t *dst, uint32_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.
 *

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

void my_memset(uint32_t value, uint8_t *dst, uint32_t length);

/********************************* my_memmove_optimized()
*****************************************
 *
 * @name  *my_memmove_optimized(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.
 *

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

void my_memmove_optimized(uint8_t *src, uint8_t *dst, uint32_t
length)__attribute__ ((optimize(3)));

/********************************* my_memset_optimized()
*********************************************
 *
 * @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.
 *

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


void my_memset_optimized(uint32_t value, uint8_t *dst, uint32_t
length)__attribute__ ((optimize(3)));

#endif /* SOURCES_MEMORY_H_ */
```

20) Nordic.c

```
/*************************************************************************
*****************************
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file nordic.c
 * @brief This file includes functions to communicate and configure
nrf module.
 * @date December 03, 2017
 *
 * long description - The nordic.c file includes functions to -
 *                      1) nrf_read_register()
 *                      2) nrf_write_register()
 *                      3) nrf_read_status()
 *                      4) nrf_read_config()
 *                      5) nrf_write_config()
 *                      6) nrf_read_rf_setup()
 *                      7) nrf_write_rf_setup()
 *                      8) nrf_read_rf_ch()
 *                      9) nrf_read_TX_ADDR()
 *                     10) nrf_write_TX_ADDR()
 *                     11) nrf_read_fifo_status()
 *                     12) nrf_flush_rx_fifo()
 *                     13) nrf_flush_tx_fifo()
 *

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

#include "nordic.h"

/********************************* nrf_read_register()
*********************************************************
 *
 * @name   -  nrf_read_register()
 * @brief  -  function to read a value from a register of nordic
module
 * @param  -  register whose value is to be read
 *
 * long description - This function reads the value from the register
and returns the value
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_register() function
definition **********************************/
```

```
uint8_t nrf_read_register(uint8_t reg)
{
      uint8_t a;
      SPI_write_byte(R_REGISTER|reg);             //sending read
command
      delay();
      SPI_read_byte();
      SPI_write_byte(0xFF);                       //sending dummy value
      delay();
      a=SPI_read_byte();                                //reading value
from register
      return a;
}

/********************************* nrf_write_register()
*********************************************************
 *
 * @name   -  nrf_write_register()
 * @brief  -  function to write a value to a register of nordic module
 * @param  -  address of the register and the value  to be written in
the register
 *
 * long description - This function writes a value to the specified
register
 *
 * @return -  void
 *

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

/********************************** nrf_write_register() function
definition *********************************/

void nrf_write_register(uint8_t reg, uint8_t value)
{
      SPI_write_byte(W_REGISTER|reg);             //sending write
command
      delay();
      SPI_read_byte();
      SPI_write_byte(value);                            //sending value
to be written
      delay();
      SPI_read_byte();
}

/********************************* nrf_read_status()
*********************************************************
 *
 * @name   -  nrf_read_status()
 * @brief  -  function to read a value of status
 *
```

```
 * long description - This function reads the value from status
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_status() function
definition *********************************/

uint8_t nrf_read_status()
{
      uint8_t a;
      nrf_chip_enable();                                    //chip enable
      a = nrf_read_register(STATUS_REG);          //read value of
status register and return value in a
      nrf_chip_disable();                                   //chip disable
      return a;
}

/********************************** nrf_read_config()
*******************************************************
 *
 * @name   -  nrf_read_config()
 * @brief  -  function to read a value of status
 *
 * long description - This function reads the value from config
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_config() function
definition *********************************/

uint8_t nrf_read_config()
{
      uint8_t a;
      nrf_chip_enable();                                    //chip enable
      a = nrf_read_register(CONFIG_REG);          //read value of
config register and return value in a
      nrf_chip_disable();
      return a;
}

/********************************** nrf_write_config()
*******************************************************
```

```
 *
 * @name   -  nrf_write_config()
 * @brief  -  function to write a value to config register
 *
 * long description - This function writes a value to config register
 *
 * @return -  void
 *

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

/********************************** nrf_read_config() function
definition *********************************/


void nrf_write_config(uint8_t config)
{
     nrf_chip_enable();                                  //chip enable
     nrf_write_register(CONFIG_REG,config);      //send config
register address and value to be written in the register
     nrf_chip_disable();                                 //chip disable
}


/********************************** nrf_read_rf_setup()
*********************************************************
 *
 * @name   -  nrf_read_rf_setup()
 * @brief  -  function to read a value of rf setup
 *
 * long description - This function reads the value from rf setup
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_rf_setup() function
definition *********************************/


uint8_t nrf_read_rf_setup()
{
     uint8_t a;
     nrf_chip_enable();                                  //chip enable
     a = nrf_read_register(RF_SETUP_REG);  //read value of rf setup
register and return value in a
     nrf_chip_disable();                                 //chip disable
     return a;
```

```
}

/********************************** nrf_write_rf_setup()
************************************************************
 *
 * @name   -  nrf_write_rf_setup()
 * @brief  -  function to write a value to rf setup register
 *
 * long description - This function writes a value to rf setup
register
 *
 * @return -  void
 *

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

/********************************** nrf_read_rf_setup() function
definition **********************************/

void nrf_write_rf_setup(uint8_t config)
{
     nrf_chip_enable();                                    //chip enable
     nrf_write_register(RF_SETUP_REG,config);//send rf setup register
address and value to be written in the register
     nrf_chip_disable();                                   //chip disable
}

/********************************** nrf_read_rf_ch()
************************************************************
 *
 * @name   -  nrf_read_rf_ch()
 * @brief  -  function to read a value of rf channel register
 *
 * long description - This function reads the value from rf channel
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_rf_ch() function
definition **********************************/

uint8_t nrf_read_rf_ch()
{
     uint8_t a;
     nrf_chip_enable();                                    //chip enable
     a = nrf_read_register(RF_CH_REG);          //read value of rf ch
register and return value in a
```

```
     nrf_chip_disable();                                //chip disable
     return a;
}

/********************************* nrf_write_rf_ch()
*********************************************************
 *
 * @name   -  nrf_write_rf_channel()
 * @brief  -  function to write a value to rf chennel register
 *
 * long description - This function writes a value to rf channel
register
 *
 * @return -  void
 *

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

/*********************************** nrf_read_rf_ch() function
definition *********************************/


void nrf_write_rf_ch(uint8_t channel)
{
     nrf_chip_enable();                                 //chip enable
     nrf_write_register(RF_CH_REG,channel);//send rf ch register
address and value to be written in the register
     nrf_chip_disable();                                //chip disable
}

/********************************* nrf_read_TX_ADDR()
*********************************************************
 *
 * @name   -  nrf_read_TX_ADDR()
 * @brief  -  function to read a value of TX ADDR register
 *
 * long description - This function reads the value from TX ADDR
register
 *
 * @return -  a(register value)
 *

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

/*********************************** nrf_read_TX_ADDR() function
definition *********************************/

void nrf_read_TX_ADDR(uint8_t *address)
{
     uint8_t i;
```

```
      nrf_chip_enable();                                    //chip enable
      SPI_write_byte(R_REGISTER|TX_ADDR);         //sending read
command in tx_addr register
      delay();
      SPI_read_byte();
      for(i=0;i<5;i++)
      {
            SPI_write_byte(0xFF);                  //sending dummy value
            delay();
            *(address+i) = SPI_read_byte();        //reading value from
register and storing it in a pointer
      }
      nrf_chip_disable();                                   //chip disable
}

/********************************* nrf_write_TX_ADDR()
*********************************************************
 *
 * @name   -  nrf_write_TX_ADDR()
 * @brief  -  function to write a value to TX ADDR register
 *
 * long description - This function writes a value to TX ADDR register
 *
 * @return -  void
 *

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

/********************************** nrf_read_TX_ADDR() function
definition *********************************/

void nrf_write_TX_ADDR(uint8_t *tx_addr)
{
      uint8_t i=0;
      nrf_chip_enable();                                    //chip enable
      SPI_write_byte(W_REGISTER|TX_ADDR);         //sending write
command in tx_addr register
      delay();
      SPI_read_byte();
      for(i=0;i<5;i++)
      {
            SPI_write_byte(*(tx_addr+i));          //writing value in
tx_addr register
            delay();
            SPI_read_byte();
      }
      nrf_chip_disable();                                   //chip disable
}

/********************************* nrf_read_fifo_status()
*********************************************************
```

```
 *
 * @name   -  nrf_read_fifo_status()
 * @brief  -  function to read a value of fifo status register
 *
 * long description - This function reads the value from fifo status
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_fifo_status() function
definition **********************************/

uint8_t nrf_read_fifo_status()
{
     uint8_t a;
     nrf_chip_enable();                                  //chip enable
     a= nrf_read_register(FIFO_STATUS_REG);//read value of fifo status
register and return value in a
     nrf_chip_disable();                                 //chip disable
     return a;
}

/********************************** nrf_flush_rx_fifo()
*********************************************************
 *
 * @name   -  nrf_flush_rx_fifo()
 * @brief  -  function to write flush_rx command
 *
 * long description - This function writes a flush_rx command
 *
 * @return -  void
 *

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

/********************************** nrf_flush_rx_fifo() function
definition **********************************/

void nrf_flush_rx_fifo()
{
     nrf_chip_enable();                                  //chip enable
     SPI_write_byte(FLUSH_RX);                    //writing flush_rx
command
     SPI_read_byte();
     nrf_chip_disable();                                 //chip disable
}
```

```
/********************************** nrf_flush_tx_fifo()
*********************************************************
 *
 * @name   -  nrf_flush_tx_fifo()
 * @brief  -  function to write flush_tx command
 *
 * long description - This function writes a flush_tx command
 *
 * @return -  void
 *

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

/********************************** nrf_flush_tx_fifo() function
definition *********************************/

void nrf_flush_tx_fifo()
{
      nrf_chip_enable();                                    //chip enable
      SPI_write_byte(FLUSH_TX);                     //writing flush_tx
command
      SPI_read_byte();
      nrf_chip_disable();                                   //chip disable
}
```

21) nordic.h

```
/*************************************************************************
*****************************
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file nordic.c
 * @brief This file includes functions to communicate and configure
nrf module.
 * @date December 03, 2017
 *
 * long description - The nordic.c file includes functions to -
 *                      1) nrf_read_register()
 *                      2) nrf_write_register()
 *                      3) nrf_read_status()
 *                      4) nrf_read_config()
 *                      5) nrf_write_config()
 *                      6) nrf_read_rf_setup()
 *                      7) nrf_write_rf_setup()
 *                      8) nrf_read_rf_ch()
 *                      9) nrf_read_TX_ADDR()
 *                     10) nrf_write_TX_ADDR()
 *                     11) nrf_read_fifo_status()
 *                     12) nrf_flush_rx_fifo()
 *                     13) nrf_flush_tx_fifo()
 *
```

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

#ifndef SOURCES_NORDIC_H_
#define SOURCES_NORDIC_H_

#include<stdio.h>
#include<stdlib.h>
#include<stdio.h>
#include "MKL25Z4.h"
#include "spi.h"
#include "gpio.h"

#define nrf_chip_enable()              (PTD_BASE_PTR->PCOR = 1)
#define nrf_chip_disable()             (PTD_BASE_PTR->PSOR = 1)
#define nrf_transmit_enable()          (PTD_BASE_PTR->PCOR = 1<<5)
#define nrf_transmit_disable()         (PTD_BASE_PTR->PSOR = 1<<5)

#define CONFIG_REG              (0x00)
#define EN_AA_REG               (0x01)
#define EN_RXADDR_REG           (0x02)
#define SETUP_AW_REG       (0x03)
#define SETUP_RETR_REG          (0x04)
#define RF_CH_REG               (0x05)
#define RF_SETUP_REG       (0x06)
#define STATUS_REG              (0x07)
#define RX_ADDR_P0_REG          (0x0A)
#define RX_ADDR_P1_REG          (0x0B)
#define TX_ADDR            (0x10)
#define FIFO_STATUS_REG    (0x17)

#define R_REGISTER              (0x00)
#define W_REGISTER             (0x20)
#define W_TX_PAYLOAD       (0xA0)
#define R_RX_PAYLOAD       (0x61)
#define FLUSH_TX          (0xE1)
#define FLUSH_RX          (0xE2)

// STATUS Register Bits
#define STATUS_RX_DR(x)      (uint8_t)((uint8_t)(x)<<6)
#define STATUS_TX_DS(x)       (uint8_t)((uint8_t)(x)<<5)
#define STATUS_MAX_RT(x)      (uint8_t)((uint8_t)(x)<<4)
#define STATUS_RX_P_NO_0    (0x00)
#define STATUS_RX_P_NO_1    (0x02)
#define STATUS_RX_P_NO_2    (0x04)
#define STATUS_RX_P_NO_3    (0x06)
#define STATUS_RX_P_NO_4    (0x08)
#define STATUS_RX_P_NO_5    (0x0A)

// CONFIG Register Bits
#define CONFIG_MASK_RX_DR(x)  (uint8_t)((uint8_t)(x)<<6)
```

```
#define CONFIG_MASK_TX_DS(x)   (uint8_t)((uint8_t)(x)<<5)
#define CONFIG_MASK_MAX_RT(x) (uint8_t)((uint8_t)(x)<<4)
#define CONFIG_EN_CRC(x)       (uint8_t)((uint8_t)(x)<<3)
#define CONFIG_CRCO_1         (0x00)
#define CONFIG_CRCO_2              (0x04)
#define CONFIG_POWER_UP       (0x02)
#define CONFIG_POWER_DOWN     (0x00)
#define CONFIG_PRIM_RX        (0x01)
#define CONFIG_PRIM_TX        (0x00)

// RF_SETUP Register Bits
#define RF_SETUP_PLL_LOCK(x)   (uint8_t)((uint8_t)(x)<<4)
#define RF_SETUP_RF_DR(x)     (uint8_t)((uint8_t)(x)<<3)
#define RF_SETUP_LNA_HCURR(x) (uint8_t)((uint8_t)(x))

// RF_CH Register Bits
#define RF_CH(x)              (uint8_t)(x);

// FIFO_STATUS Register Bits
#define FIFO_STATUS_TX_REUSE   (FIFO_STATUS_REG &
(uint8_t)((uint8_t)(1)<<6))
#define FIFO_STATUS_TX_FULL    (FIFO_STATUS_REG &
(uint8_t)((uint8_t)(1)<<5))
#define FIFO_STATUS_TX_EMPTY   (FIFO_STATUS_REG &
(uint8_t)((uint8_t)(1)<<4))
#define FIFO_STATUS_RX_FULL    (FIFO_STATUS_REG &
(uint8_t)((uint8_t)(1)<<1))
#define FIFO_STATUS_RX_EMPTY   (FIFO_STATUS_REG &
(uint8_t)((uint8_t)(1)<<0))

/********************************** nrf_read_register()
*********************************************************
 *
 * @name   -  nrf_read_register()
 * @brief  -  function to read a value from a register of nordic
module
 * @param  -  register whose value is to be read
 *
 * long description - This function reads the value from the register
and returns the value
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_register() function
definition *********************************/

uint8_t nrf_read_register(uint8_t reg);
```

```
 /********************************** nrf_write_register()
*************************************************************
  *
  * @name   -  nrf_write_register()
  * @brief  -  function to write a value to a register of nordic
module
  * @param  -  address of the register and the value  to be written in
the register
  *
  * long description - This function writes a value to the specified
register
  *
  * @return -  void
  *

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

 /*********************************** nrf_write_register() function
definition ***********************************/

void nrf_write_register(uint8_t reg, uint8_t value);

/********************************** nrf_read_status()
*************************************************************
 *
 * @name   -  nrf_read_status()
 * @brief  -  function to read a value of status
 *
 * long description - This function reads the value from status
register
 *
 * @return -  a(register value)
 *

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

/*********************************** nrf_read_status() function
definition ***********************************/

uint8_t nrf_read_status();

/********************************** nrf_read_config()
*************************************************************
 *
 * @name   -  nrf_read_config()
 * @brief  -  function to read a value of status
 *
 * long description - This function reads the value from config
register
 *
```

```
 * @return -  a(register value)
 *

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

/*********************************** nrf_read_config() function
definition **********************************/

uint8_t nrf_read_config();

/********************************** nrf_write_config()
*******************************************************
 *
 * @name   -  nrf_write_config()
 * @brief  -  function to write a value to config register
 *
 * long description - This function writes a value to config register
 *
 * @return -  void
 *

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

/*********************************** nrf_read_config() function
definition **********************************/

void nrf_write_config(uint8_t config);

/********************************** nrf_read_rf_setup()
*******************************************************
 *
 * @name   -  nrf_read_rf_setup()
 * @brief  -  function to read a value of rf setup
 *
 * long description - This function reads the value from rf setup
register
 *
 * @return -  a(register value)
 *

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

/*********************************** nrf_read_rf_setup() function
definition **********************************/

uint8_t nrf_read_rf_setup();

/********************************** nrf_write_rf_setup()
*******************************************************
```

```
 *
 * @name   -  nrf_write_rf_setup()
 * @brief  -  function to write a value to rf setup register
 *
 * long description - This function writes a value to rf setup
register
 *
 * @return -  void
 *

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

/********************************** nrf_read_rf_setup() function
definition **********************************/

void nrf_write_rf_setup(uint8_t config);

/********************************** nrf_read_rf_ch()
*************************************************************
 *
 * @name   -  nrf_read_rf_ch()
 * @brief  -  function to read a value of rf channel register
 *
 * long description - This function reads the value from rf channel
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_rf_ch() function
definition **********************************/

uint8_t nrf_read_rf_ch();

/********************************** nrf_write_rf_ch()
*************************************************************
 *
 * @name   -  nrf_write_rf_channel()
 * @brief  -  function to write a value to rf chennel register
 *
 * long description - This function writes a value to rf channel
register
 *
 * @return -  void
 *

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

```
/********************************** nrf_read_rf_ch() function
definition **********************************/

void nrf_write_rf_ch(uint8_t channel);

/********************************** nrf_read_TX_ADDR()
*****************************************************
 *
 * @name   -  nrf_read_TX_ADDR()
 * @brief  -  function to read a value of TX ADDR register
 *
 * long description - This function reads the value from TX ADDR
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_TX_ADDR() function
definition **********************************/

void nrf_read_TX_ADDR(uint8_t *address);

/********************************** nrf_write_TX_ADDR()
*****************************************************
 *
 * @name   -  nrf_write_TX_ADDR()
 * @brief  -  function to write a value to TX ADDR register
 *
 * long description - This function writes a value to TX ADDR register
 *
 * @return -  void
 *

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

/********************************** nrf_read_TX_ADDR() function
definition **********************************/

void nrf_write_TX_ADDR(uint8_t *tx_addr);

/********************************** nrf_read_fifo_status()
*****************************************************
 *
 * @name   -  nrf_read_fifo_status()
 * @brief  -  function to read a value of fifo status register
 *
```

```
 * long description - This function reads the value from fifo status
register
 *
 * @return -  a(register value)
 *

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

/********************************** nrf_read_fifo_status() function
definition *********************************/

uint8_t nrf_read_fifo_status();

/********************************** nrf_flush_rx_fifo()
*********************************************************
 *
 * @name   -  nrf_flush_rx_fifo()
 * @brief  -  function to write flush_rx command
 *
 * long description - This function writes a flush_rx command
 *
 * @return -  void
 *

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

/********************************** nrf_flush_rx_fifo() function
definition *********************************/

void nrf_flush_tx_fifo();

/********************************** nrf_flush_tx_fifo()
*********************************************************
 *
 * @name   -  nrf_flush_tx_fifo()
 * @brief  -  function to write flush_tx command
 *
 * long description - This function writes a flush_tx command
 *
 * @return -  void
 *

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

/********************************** nrf_flush_tx_fifo() function
definition *********************************/

void nrf_flush_rx_fifo();
```

```
#endif /* SOURCES_NORDIC_H_ */
```

22) profiler.c

```
/*****************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file profiler.c
 * @brief This file includes functions for profiling using Systick
Timer.
 * @date November 29, 2017
 *
 * long description - The profiler.c file includes functions to -
 *                      1) configure the Systick profiler
(profiler_start())
 *                      2) disable profiling (profiler_stop())
 *                      3) get systick val (gettime())
 *                      4) calculate execution time (execution_time())
 *                      5) handle Systick interupt (SysTick_Handler())
 *
*****************************************************************************
****************************/

#include "profiler.h"
#include "project3.h"


/********************************** profiler_start()
*********************************************************
 *
 * @name   -  profiler_start()
 * @brief  -  function to configure the Systick profiler.
 * @param  -  none
 *
 * long description - This function configures the Systick profiler.
 *
 * @return -  void
 *

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

/********************************** profiler_start() function
definition *********************************/
void profiler_start()
{
      SysTick->LOAD = SYSTICK_MAX_VALUE;
      SysTick->VAL = 5;
      SysTick->CTRL = 0x00000007;           //start timer
```

```
      overflow=0;
}

/********************************** profiler_stop()
************************************************************
 *
 * @name   -  profiler_stop()
 * @brief  -  function to disable profiling
 * @param  -  none
 *
 * long description - This function disables profiling.
 *
 * @return -  void
 *

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

/*********************************** profiler_stop() function
definition **********************************/

void profiler_stop()
{
      SysTick->CTRL = 0x00;                  //stop timer
}


/********************************** gettime()
************************************************************
 *
 * @name   -  gettime()
 * @brief  -  function to get systick val.
 * @param  -  none
 *
 * long description - This function gets systick val
 *
 * @return -  systick value
 *

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

/*********************************** gettime() function definition
**********************************/

volatile uint32_t gettime()
{
      return SysTick->VAL;
}
```

```
/********************************** execution_time()
*********************************************************
 *
 * @name   -  execution_time()
 * @brief  -  function to calculate execution time.
 * @param  -  start_time:systick val before code execution
 * @param  -  end_time:systick val after code execution
 *
 * long description - This function calculates execution time.
 *
 * @return -  execution time
 *

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

/********************************** execution_time() function
definition *********************************/

volatile uint32_t execution_time(uint32_t start_time,uint32_t
end_time)
{
      uint32_t start = SYSTICK_MAX_VALUE - start_time;
      uint32_t end = (SYSTICK_MAX_VALUE -
end_time)+(overflow*SYSTICK_MAX_VALUE);
      uint32_t clocks = end - start;
      uint8_t time1 = clocks/48;
      return time1;
}

/********************************** SysTick_Handler()
*********************************************************
 *
 * @name   -  SysTick_Handler()
 * @brief  -  function to handle Systick interupt
 * @param  -  none
 *
 * long description - This function handles Systick interrupt.
 *
 * @return -  void
 *

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

 /********************************** SysTick_Handler() function
definition *********************************/

void SysTick_Handler(void)
{
      overflow++;
      if(TPM2_SC & TPM_SC_TOF_MASK)
```

```
      {
            TPM2->CNT = 10;                                          // clear
count
            TPM2_SC &= ~TPM_SC_TOF_MASK;                  // clear timer
overflow flag
      }
}
```

23) profiler.h

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file profiler.c
 * @brief This file includes functions for profiling using Systick
Timer.
 * @date November 29, 2017
 *
 * long description - The profiler.c file includes functions to -
 *                      1) configure the Systick profiler
(profiler_start())
 *                      2) disable profiling (profiler_stop())
 *                      3) get systick val (gettime())
 *                      4) calculate execution time (execution_time())
 *                      5) handle Systick interupt (SysTick_Handler())
 *

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

#ifndef SOURCES_PROFILER_H_
#define SOURCES_PROFILER_H_

#include <stdint.h>
#include "MKL25Z4.h"
#include <stdlib.h>
#include "project3.h"

#define SYSTICK_MAX_VALUE 0x00FFFFFFU


uint32_t overflow;

/********************************** profiler_start()
*********************************************************
 *
 * @name   -  profiler_start()
 * @brief  -  function to configure the Systick profiler.
 * @param  -  none
 *
 * long description - This function configures the Systick profiler.
```

```
 *
 * @return -  void
 *

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

void profiler_start();

/********************************* profiler_stop()
*********************************************************
 *
 * @name   -  profiler_stop()
 * @brief  -  function to disable profiling
 * @param  -  none
 *
 * long description - This function disables profiling.
 *
 * @return -  void
 *

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

void profiler_stop();

/********************************* gettime()
*********************************************************
 *
 * @name   -  gettime()
 * @brief  -  function to get systick val.
 * @param  -  none
 *
 * long description - This function gets systick val
 *
 * @return -  systick value
 *

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

volatile uint32_t gettime();

/********************************* execution_time()
*********************************************************
 *
 * @name   -  execution_time()
 * @brief  -  function to calculate execution time.
 * @param  -  start_time:systick val before code execution
 * @param  -  end_time:systick val after code execution
 *
 * long description - This function calculates execution time.
```

```
 *
 * @return -  execution time
 *

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

volatile uint32_t execution_time(uint32_t start_time,uint32_t
end_time);

/********************************* SysTick_Handler()
*************************************************************
 *
 * @name   -  SysTick_Handler()
 * @brief  -  function to handle Systick interupt
 * @param  -  none
 *
 * long description - This function handles Systick interrupt.
 *
 * @return -  void
 *

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

void SysTick_Handler(void);

#endif /* SOURCES_PROFILER_H_ */
```

24) rtc.c

```
/*****************************************************************************
*****************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file rtc.c
 * @brief This source file includes RTC functions
 * @date November 29, 2017
 *
 * long description - The rtc.c file includes functions to -
 *                       1) initialize RTC (RTC_init())
 *                       2) handle RTC_Seconds Interrupt
(RTC_Seconds_IRQHandler())
 *                       3) read RTC Timer Seconds Register
(RTC_read())
 *
 *

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

#include "rtc.h"
```

```
#include "logger.h"
#include "project3.h"

/********************************** RTC_init()
*****************************************************
 *
 * @name   -  RTC_init()
 * @brief  -  function to initialize RTC
 * @param  -  none
 *
 * long description - This function configures RTC control and status
registers, sets clock,oscillator,
 *                    and clock pins, enables seconds interrupt,
selects clock mode.
 *
 * @return -  void
 *

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

/********************************* RTC_init() function definition
***********************************/

void RTC_init(void)
{
     SIM->SCGC6 |= SIM_SCGC6_RTC(1);                  // Enable RTC
clock gate
     SIM->SCGC5 |= SIM_SCGC5_PORTC(1);                // Enable PORTC
clock gate

     MCG_C1 |= MCG_C1_IRCLKEN_MASK;                   //Enable
internal reference clock
     MCG_C2 &= ~(MCG_C2_IRCS_MASK);              //Internal Reference
Clock -->Slow

     PORTC_PCR1 |= (PORT_PCR_MUX(1));            // Configure PTC1 as
Clock input pin
     PORTC_PCR3 |= (PORT_PCR_MUX(0x5));               //Configure PTC3
as Clock output pin
     SIM_SOPT2 &= ~(SIM_SOPT2_CLKOUTSEL_MASK);   // Clear CLKOUTSEL
bit field
     SIM_SOPT2 |= SIM_SOPT2_CLKOUTSEL(4);        // Set CLKOUTSEL bit
field
     SIM_SOPT1 &= ~(SIM_SOPT1_OSC32KSEL_MASK);   // Clear OSC32KSEL
bit field
     SIM_SOPT1 |= SIM_SOPT1_OSC32KSEL(2);        //PTC3 as CLKOUT

     if (RTC_SR & RTC_SR_TIF_MASK){
          RTC_TSR = 0;                                    //clears
the TIF
     }
```

```
      RTC_SR |= RTC_SR_TCE(1);
      RTC_IER &= ~RTC_IER_TSIE(1);                        //Seconds
interrupt disable

      NVIC_SetPriority(RTC_Seconds_IRQn,21);      // Setting
RTC_Seconds interrupt priority
      NVIC_ClearPendingIRQ(RTC_Seconds_IRQn);          // Clearing
pending RTC_Seconds interrupt (if any)
      NVIC_EnableIRQ(RTC_Seconds_IRQn);                // Enabling
RTC_Seconds interrupt

}

/********************************** RTC_Seconds_IRQHandler()
*****************************************************
 *
 * @name   -  RTC_Seconds_IRQHandler()
 * @brief  -  function to handle RTC seconds interrupt
 * @param  -  none
 *
 * long description - This function handles RTC seconds interrupt and
logs HEARTBEAT after every second.
 *
 * @return -  void
 *

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

/********************************** RTC_Seconds_IRQHandler() function
definition ***********************************/


void RTC_Seconds_IRQHandler(void){
      LOG(HEARTBEAT,NULL);
}

/********************************** RTC_read()
*****************************************************
 *
 * @name   -  RTC_read()
 * @brief  -  function to read RTC Time Seconds Register
 * @param  -  none
 *
 * long description - This function read RTC Time Seconds Register.
 *
 * @return -  RTC->TSR value
 *

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

```
/********************************* RTC_read() function definition
**********************************/

uint32_t RTC_read(void){
     return RTC->TSR;
}
```

25) rtc.h

```
/*************************************************************************
*****************************
 *
 * @author Shivam Khandelwal, Preshit Harlikar
 * @file rtc.h
 * @brief This header file includes RTC function declarations
 * @date November 29, 2017
 *
 * long description - The rtc.h file includes function declarations of
functions to -
 *                      1) initialize RTC (RTC_init())
 *                      2) handle RTC_Seconds Interrupt
(RTC_Seconds_IRQHandler())
 *                      3) read RTC Timer Seconds Register
(RTC_read())
 *
 *

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

#ifndef SOURCES_RTC_H_
#define SOURCES_RTC_H_

#include <stdint.h>
#include "MKL25Z4.h"
#include "project3.h"


/********************************* RTC_init()
*****************************************************
 *
 * @name   -  RTC_init()
 * @brief  -  function to initialize RTC
 * @param  -  none
 *
 * long description - This function configures RTC control and status
registers, sets clock,oscillator,
 *                    and clock pins, enables seconds interrupt,
selects clock mode.
 *
 * @return -  void
```

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

/********************************** RTC_init() function declaration
***********************************/

void RTC_init(void);

/********************************** RTC_Seconds_IRQHandler()
*********************************************************
 *
 * @name   -  RTC_Seconds_IRQHandler()
 * @brief  -  function to handle RTC seconds interrupt
 * @param  -  none
 *
 * long description - This function handles RTC seconds interrupt and
logs HEARTBEAT after every second.
 *
 * @return -  void
 *

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

/********************************** RTC_Seconds_IRQHandler() function
declaration ***********************************/

void RTC_Seconds_IRQHandler(void);

/********************************** RTC_read()
*********************************************************
 *
 * @name   -  RTC_read()
 * @brief  -  function to read RTC Time Seconds Register
 * @param  -  none
 *
 * long description - This function read RTC Time Seconds Register.
 *
 * @return -  RTC->TSR value
 *

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

/********************************** RTC_read() function declaration
***********************************/

uint32_t RTC_read(void);
```

```
#endif /* SOURCES_RTC_H_ */
```

26) spi.c

```
/*****************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file spi.c
 * @brief This file includes Circular Buffer functions
 * @date November 26, 2017
 *
 * long description - The spi.c file includes functions to -
 *                       1) SPI initialization function SPI_init()
 *                       2) SPI status function SPI_status()
 *                       3) SPI write function SPI_write() to write one
byte data
 *                       4) SPI read function SPI_write() to read one
byte data
 *                       5) SPI send packet function SPI_send_packet()
to write n bytes of data
 *                       6) SPI_flush() function to disable SPI
 *

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


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

#include "spi.h"
#include "gpio.h"

/********************************* SPI_init()
*************************************************
 *
 * @name   -  SPI_init()
 * @brief  -  function to initialize SPI
 *
 * long description - This function initializes SPI i.e sets baud rate
and enables SPI transfer
 *

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

void SPI_init(void)
{

     SIM->SCGC4 |= SIM_SCGC4_SPI0_MASK;                    //enable
clock for SPI_0
```

```
      SPI0->BR = (SPI_BR_SPPR(0x01) | SPI_BR_SPR(0x00));     //baud
rate

      SPI0->C1 = 0x50;
}


/********************************** SPI_status()
***********************************************
 *
 * @name   -  SPI_status()
 * @brief  -  function to read status of SPI
 *
 * long description - This function reads SPI status register and
returns the value
 *
 * @return -  SPI0->S : value of SPI status register
 *

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

uint8_t SPI_status()
{
      return SPI0->S;
      //return SPI status
}

/********************************** SPI_write_byte(uint8_t byte)
*********************************
 *
 * @name   -  SPI_write_byte(uint8_t byte)
 * @brief  -  function to write one byte of data
 *
 * long description - This function takes in the byte to be written
and transfers it through SPI
 *

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


void SPI_write_byte(uint8_t byte)
{
      while (!(SPI0->S & 0x20));
      //check whether transfer buffer is empty
      SPI0->D = byte;
      //write data
}
```

```
/********************************** SPI_read_byte()
************************************************
 *
 * @name   -  SPI_read_byte()
 * @brief  -  function reads byte in SPI data register after receiving
is completed
 *
 * long description - This function reads SPI data register and
returns the value
 *
 * @return -  byte : value of SPI data register
 *

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

uint8_t SPI_read_byte()
{
      uint8_t byte;
      while (!(SPI0->S & 0x80));
      //check whether receive buffer is full
      byte = SPI0->D;
      //read data
      return byte;
      //return data
}

/****************************SPI_send_packet(uint8_t *p, uint8_t
length) ***************************
 *
 * @name   -  SPI_send_packet(uint8_t *p, uint8_t length)
 * @brief  -  function to write n bytes of data to SPI
 *
 * long description - This function takes in the length and pointer to
the data to be written and
 *                       transfers it through SPI
 *

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

void SPI_send_packet(uint8_t *p, uint8_t length)
{
      uint8_t i = 0;
      for(i=0;i<length;i++)
      {
            while (!(SPI0->S & 0x20));
      //check whether transfer buffer is empty
            SPI0->D = *(p+i);
      //write data
      }
}
```

```
/********************************** SPI_flush()
**************************************************
 *
 * @name   -  SPI_flush()
 * @brief  -  function to disable SPI and reinitialize it
 *
 * long description - This function disable SPI module and
reinitializes SPI
 *

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

void SPI_flush()
{
      SPI0->C1= SPI0->C1 & 0xBF;                                    //SPI
module disable
      SPI_init();
      //reinitialize SPI
}

/********************************** delay()
**************************************************
 *
 * @name   -  delay()
 * @brief  -  function to set delay
 *
 * long description - This function is used to set a delay
 *

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

void delay()
{
      uint32_t i=0;
      for(i=0;i<1000;i++);
}
```

27) spi.h

```
/*****************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file spi.h
 * @brief This file includes Circular Buffer functions
 * @date November 26, 2017
 *
 * long description - The spi.c file includes functions to -
```

```
 *                      1) SPI initialization function SPI_init()
 *                      2) SPI status function SPI_status()
 *                      3) SPI write function SPI_write() to write one
byte data
 *                      4) SPI read function SPI_write() to read one
byte data
 *                      5) SPI send packet function SPI_send_packet()
to write n bytes of data
 *                      6) SPI_flush() function to disable SPI
 *

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

#ifndef SOURCES_SPI_H_
#define SOURCES_SPI_H_

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

#include<stdio.h>
#include<stdlib.h>
#include<stdint.h>
#include "MKL25Z4.h"
#include "gpio.h"

/********************************** SPI_init()
*****************************************************
 *
 * @name   -  SPI_init()
 * @brief  -  function to initialize SPI
 *
 * long description - This function initializes SPI i.e sets baud rate
and enables SPI transfer
 *

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

void SPI_init();

/********************************** SPI_status()
*************************************************
 *
 * @name   -  SPI_status()
 * @brief  -  function to read status of SPI
 *
 * long description - This function reads SPI status register and
returns the value
 *
 * @return -  SPI0->S : value of SPI status register
 *
```

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

uint8_t SPI_status();

/********************************* SPI_write_byte(uint8_t byte)
*********************************
 *
 * @name   -  SPI_write_byte(uint8_t byte)
 * @brief  -  function to write one byte of data
 *
 * long description - This function takes in the byte to be written
and transfers it through SPI
 *

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

void SPI_write_byte(uint8_t byte);

/********************************* SPI_read_byte()
************************************************
 *
 * @name   -  SPI_read_byte()
 * @brief  -  function reads byte in SPI data register after receiving
is completed
 *
 * long description - This function reads SPI data register and
returns the value
 *
 * @return -  byte : value of SPI data register
 *

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

uint8_t SPI_read_byte();

/****************************SPI_send_packet(uint8_t *p, uint8_t
length) ***************************
 *
 * @name   -  SPI_send_packet(uint8_t *p, uint8_t length)
 * @brief  -  function to write n bytes of data to SPI
 *
 * long description - This function takes in the length and pointer to
the data to be written and
 *                      transfers it through SPI
 *

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

```
void SPI_send_packet(uint8_t *p, uint8_t length);

/********************************** SPI_flush()
*************************************************
 *
 * @name   -  SPI_flush()
 * @brief  -  function to disable SPI and reinitialize it
 *
 * long description - This function disable SPI module and
reinitializes SPI
 *

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


void SPI_flush();

/********************************** delay()
*************************************************
 *
 * @name   -  delay()
 * @brief  -  function to set delay
 *
 * long description - This function is used to set a delay
 *

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

void delay();

#endif /* SOURCES_SPI_H_ */
```

28) uart.c

```
/*
 * uart.c
 *
 *  Created on: 03-Dec-2017
 *      Author: defaultuser0
 */
/*****************************************************************************
*****************************
 *
 * @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"



uint8_t log_mode;
/******************************** 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
*********************************/

UART_status UART_configure()
{
      SIM->SCGC5 |= SIM_SCGC5_PORTA_MASK;
      /* Enable GPIOA Clock */
      SIM->SCGC4 |= SIM_SCGC4_UART0_MASK;
            /* Enable UART0 Clock */
      SIM->SCGC5 |= SIM_SCGC5_PORTB_MASK;
```

```
	PORTA_PCR1 |= PORT_PCR_MUX(2);
		/* Configuring PCR1 to alt2 for UART0 */
	PORTA_PCR2 |= PORT_PCR_MUX(2);



	/* Configuring PCR2 to alt2 for UART0 */

	SIM->SOPT2 &= ~(SIM_SOPT2_PLLFLLSEL_MASK);
	SIM->SOPT2 |= SIM_SOPT2_PLLFLLSEL(1);
	/* Selecting PLL clock mode */
	SIM->SOPT2 |= SIM_SOPT2_UART0SRC(1);
	/* Using MCGFLLPLL */

	uint16_t baud_rate = BAUD_RATE;
	/* Setting baud rate registers */
	UART0_BDL = (uint8_t)(baud_rate & UART0_BDL_SBR_MASK) ;
	UART0_BDH = (uint8_t)((baud_rate>>8) & UART0_BDH_SBR_MASK);

	UART0_C1 = 0x00;
		/* Configuring C1 control register */
	UART0_C3 = 0x00;
			/* Configuring C3 control register */
	UART0_S1 = 0x00;
			/* Configuring S1 status register */
	UART0_C4 = UART0_C4_OSR(15);
		/* Setting OSR bit field to 16 */

	UART0_C2 &= ~(UART0_C2_RE_MASK | UART0_C2_TE_MASK);
	/* Clear TE and RE */
	UART0_C2 |=  (UART0_C2_RE_MASK | UART0_C2_TE_MASK);
	/* Set TE and RE */
	UART0_C2 |= UART0_C2_RIE_MASK;
		/* Set RIE */
	//UART0_C2 &= ~UART0_C2_TIE_MASK;
	//UART0_C2 |= UART0_C2_TIE_MASK;

	SIM_BASE_PTR->SCGC5 |= SIM_SCGC5_PORTB_MASK;
	SIM_BASE_PTR->SCGC5 |= SIM_SCGC5_PORTD_MASK;

	//if(log_mode == 0x43){
	NVIC_SetPriority(UART0_IRQn,12);
	//NVIC_CLearPendingIRQ(UART0_IRQn);
	NVIC_EnableIRQ(UART0_IRQn);
	//}                                                    /* Enable UART0
Interrupt */

	return UART_CONFIG_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
*********************************/

UART_status UART_send(uint8_t *data0)
{
           while(!(UART0_S1 & UART0_S1_TDRE_MASK));
           UART0_D = *data0;
                                                                    /*
Writing data to UART0 data register */
     return TX_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
*********************************/

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

```
	if(log_mode == 0x43){
		for(i=0;i<length;i++){
			UART0_D = *(data0 + i);
				/* Writing data to UART0 data register */
		}
	}
	else{
	for(i=0;i<length;i++)
	{
		while(!(UART0_S1 & UART0_S1_TDRE_MASK));
		UART0_D = *(data0 + i);
			/* Writing data to UART0 data register */
	}
	}
	return TX_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
*********************************/

UART_status UART_receive(uint8_t *data0)
{
	if(log_mode == 0x43){
		*(data0) = UART0_D;
	}
	else{
		while(!(UART0_S1 & UART0_S1_RDRF_MASK));
		*(data0) = UART0_D;
				/* Reading data from UART0 data register */
	}
	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
**********************************/

UART_status UART_receive_n(uint8_t *data0, uint8_t length)
{
      uint8_t i = 0;
      if(log_mode == 0x43){
            for(i=0;i<length;i++){
                  *(data0 + i) = UART0_D;
      /* Reading data from UART0 data register */
            }
      }
      else{
            for(i=0;i<length;i++){
                  while(!(UART0_S1 & UART0_S1_RDRF_MASK));
                  *(data0 + i) = UART0_D;
      /* Reading data from UART0 data register */
            }
      }
      /* Reading data from UART0 data register */
      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 *********************************/

UART_status UART0_IRQHandler(void)
{
     //if((UART0_S1 & UART0_S1_TDRE_MASK) &&
(!CB_is_empty(&Tx_Buffer))){ /* Check if TDRE set and Transmit buffer
is not empty */
     //    CB_buffer_remove_item(&Tx_Buffer, &Tx_Data);
        /* Write data from Tx_Buffer to Tx_data */
     //    UART_send(&Tx_Data);
             /* Send data from Tx_Data */
     //    if(CB_is_empty(&Tx_Buffer)){
             /* Check if Transmit buffer is empty */
     //         UART0_C2 &= ~UART0_C2_TIE_MASK;
             /* Clear TIE bit */
     //    }
     //    return TX_IRQ;
     //}

     //__disable_irq();


     if((UART0_S1 & UART0_S1_RDRF_MASK ) &&
(!CB_is_full(&Rx_Buffer))){ /* Check if RDRF set and Receive buffer is
not full */
           START_CRITICAL();
           UART_receive(&Rx_Data);
                    /* Write received data to Rx_Data */
           CB_buffer_add_item(&Rx_Buffer, &Rx_Data);
         /* Add data from Rx_Data to Rx_Buffer */
           if(CB_is_full(&Rx_Buffer)){
              /* Check if Receive buffer is full */
                UART0_C2 &= ~UART0_C2_RIE_MASK;
              /* Clear RIE bit */
           }
           END_CRITICAL();
           return RX_IRQ;
     }

     if((UART0_S1 & UART0_S1_TDRE_MASK) &&
(!(Log_buffer_is_empty(&log_buffer)))){ /* Check if TDRE set and
Transmit buffer is not empty */
           START_CRITICAL();
                Log_buffer_remove_item(&log_buffer, &Log_Tx_Data);
```

```
                //log_mode = 0x43;/* Write data from Tx_Buffer to
Tx_data */
                UART_send(&Log_Tx_Data);
                        /* Send data from Tx_Data */
                if(Log_buffer_is_empty(&log_buffer) ==
LOG_BUFFER_EMPTY){                                              /*
Check if Transmit buffer is empty */
                     UART0_C2 &= ~UART0_C2_TIE_MASK;
                   /* Clear TIE bit */
                }
           END_CRITICAL();
           return TX_IRQ;
     }
     else{
           return UART_CONFIG_SUCCESS;

     }
     //__enable_irq();
}
```

29) uart.h

```
/*
 * uart.h
 *
 *  Created on: 03-Dec-2017
 *      Author: defaultuser0
 */

#ifndef SOURCES_UART_H_
#define SOURCES_UART_H_

#include <stdint.h>
#include "MKL25Z4.h"
#include <stdlib.h>
#include "circbuf.h"
#include "project3.h"

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



typedef enum{
     TX_SUCCESS=6,
     RX_SUCCESS=7,
     TX_IRQ=8,
     RX_IRQ=9,
     UART_CONFIG_SUCCESS
}UART_status;
```

```
extern uint8_t log_mode;

UART_status UART_configure();
UART_status UART_send(uint8_t *data0);
UART_status UART_send_n(uint8_t *data0, uint8_t length);
UART_status UART_receive(uint8_t *data0);
UART_status UART_receive_n(uint8_t *data0, uint8_t length);
UART_status UART0_IRQHandler(void);

#endif /* SOURCES_UART_H_ */
```

30) test.c

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file test.c
 * @brief This file includes functions for testing profiling, spi and
nrf
 * @date November 29, 2017
 *
 * long description - The test.c file includes functions to -
 *                      1) test profile time for memory functions
dma,non-dma,stdlib) (profile_test())
 *                      2) test read and write operation to nrf module
(spi_nrf_test())
 *                      3) test config read and write operation to nrf
module (config_reg_test())
 *                      4) test read and write operation to nrf module
(tx_addr_test())
 *                      5) test read and write operation to nrf module
(rf_setup_test())
 *                      6) test read and write operation to nrf module
(rf_ch_reg_test())
 *                      7) read the status register of nrf module
(status_reg_test())
 *                      8) read the status register of nrf module
(fifo_status_reg_test())
 *                      9) test and log on UART the success of read
and write operations for nrf registers (transmit())
 *                     10) test profiling time of memmove (stdlib)
function (lib_memmove_test())
 *                     11) test profiling time of memmove (non-dma)
function (my_memmove_test())
 *                     12) test profiling time of memmove (dma)
function (dma_memmove_test())
 *                     13) test profiling time of memset (dma)
function (dma_memset_test())
```

```
 *                      14) test profiling time of memset (non-dma)
function (my_memset_test())
 *                      15) test profiling time of memset (stdlib)
function (lib_memset_test())
 *

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

#include "test.h"

/*********************************
profile_test()*************************************************
**
 *
 * @name   -  profile_test()
 * @brief  -  function to test profile time for memory functions
dma,non-dma,stdlib)
 * @param  -  none
 *
 * long description - This function passes log struct to log buffer.
 *
 * @return -  void
 *

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

/********************************** profile_test() function
definition **********************************/

void profile_test()
{
      uart_flush("\n\r\n\r");
      uart_flush("Number of bytes  ");
      log_int(10);
      log_int(100);
      log_int(1000);
      log_int(5000);
      uart_flush("\n\r");
      uart_flush("lib-memmove       ");
      lib_memmove_test(10);
      log_int(t);
      lib_memmove_test(100);
      log_int(t);
      lib_memmove_test(1000);
      log_int(t);
      lib_memmove_test(5000);
      log_int(t);
      uart_flush("\n\r");
      uart_flush("my-memmove        ");
      my_memmove_test(10);
```

```
log_int(t);
my_memmove_test(100);
log_int(t);
my_memmove_test(1000);
log_int(t);
my_memmove_test(5000);
log_int(t);
uart_flush("\n\r");
uart_flush("my-memmove-O3  ");
my_memmove_test_optimized(10);
log_int(t);
my_memmove_test_optimized(100);
log_int(t);
my_memmove_test_optimized(1000);
log_int(t);
my_memmove_test_optimized(5000);
log_int(t);
uart_flush("\n\r");
uart_flush("dma-memmove       ");
dma_memmove_test(10);
log_int(t);
dma_memmove_test(100);
log_int(t);
dma_memmove_test(1000);
log_int(t);
dma_memmove_test(5000);
log_int(t);
uart_flush("\n\r\n\r");
uart_flush("Number of bytes  ");
log_int(10);
log_int(100);
log_int(1000);
log_int(5000);
uart_flush("\n\r");
uart_flush("lib-memset        ");
lib_memset_test(10);
log_int(t);
lib_memset_test(100);
log_int(t);
lib_memset_test(1000);
log_int(t);
lib_memset_test(5000);
log_int(t);
uart_flush("\n\r");
uart_flush("my-memset       ");
my_memset_test(10);
log_int(t);
my_memset_test(100);
log_int(t);
my_memset_test(1000);
log_int(t);
my_memset_test(5000);
```

```
      log_int(t);
      uart_flush("\n\r");
      uart_flush("my-memset-O3    ");
      my_memset_test_optimized(10);
      log_int(t);
      my_memset_test_optimized(100);
      log_int(t);
      my_memset_test_optimized(1000);
      log_int(t);
      my_memset_test_optimized(5000);
      log_int(t);
      uart_flush("\n\r");
      uart_flush("dma-memset        ");
      dma_memset_test(10);
      log_int(t);
      dma_memset_test(100);
      log_int(t);
      dma_memset_test(1000);
      log_int(t);
      dma_memset_test(5000);
      log_int(t);
      uart_flush("\n\r\n\r");
      Rxd_Data=0;
}

/*********************************
spi_nrf_test()()*************************************************************
****
 *
 * @name   -  spi_nrf_test()
 * @brief  -  function to test read and write operation to nrf module
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

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

/********************************** spi_nrf_test() function
definition **********************************/

void spi_nrf_test()
{
      SPI_init();
      //initialize SPI
      tx_addr_test();                                    //test function
for tx_addr register
```

```
      rf_setup_test();                                  //test function for
rf_setup register
      rf_ch_reg_test();                                       //test function
for rf_ch register
      config_reg_test();                                      //test function
for config register
      status_reg_test();                                      //test function
for status register
      fifo_status_reg_test();                                 //test function
for fifo_status register
      uart_flush("\n\r\n\r");
      Rxd_Data=0;
}

/*********************************
config_reg_test()*****************************************************
*****
 *
 * @name   -  config_reg_test()
 * @brief  -  function to test config read and write operation to nrf
module.
 * @param  -  none
 * long description - This function tests config read and write
operation to nrf module.
 *
 * @return -  void
 *

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

/********************************** config_reg_test() function
definition **********************************/

void config_reg_test()
{
      nrf_write_config(0x74);
      uint8_t config;
      config = nrf_read_config();
      delay();
      transmit(CONFIG);
      if(config==0x74){
            uart_flush("SUCCESSFUL");
      }
      else{
            uart_flush("FAILED");
      }
      SPI_flush();
}
```

```
/*********************************
tx_addr_test()*************************************************************
**
 *
 * @name   -  tx_addr_test()
 * @brief  -  function to test read and write operation to nrf module.
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

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

/********************************** tx_addr_test() function
definition *********************************/

void tx_addr_test()
{
      uint8_t write[5]={0xAA,0x99,0x88,0x77,0x66};
      nrf_write_TX_ADDR(write);
      uint8_t read[5];
      nrf_read_TX_ADDR(read);
      delay();
      uint8_t i=0;
      uint8_t j=0;
      transmit(TXADDR);
      while(i<5)
      {
            if(*(read+i)==*(write+i))
            {
                  j++;
            }
            i++;
      }
      if(j==5){
            uart_flush("SUCCESSFUL");
      }
      else{
            uart_flush("FAILED");
      }
      SPI_flush();
}


/*********************************
rf_setup_test()************************************************************
***
 *
```

```
 * @name   -  rf_setup_test()
 * @brief  -  function to test read and write operation to nrf module.
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module.
 *
 * @return -  void
 *

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

/********************************** rf_setup_test() function
definition *********************************/

void rf_setup_test()
{
      nrf_write_rf_setup(0x09);
      uint8_t setup;
      setup = nrf_read_rf_setup();
      delay();
      transmit(RFSETUP);
      if(setup==0x09){
            uart_flush("SUCCESSFUL");
      }
      else{
            uart_flush("FAILED");
      }
      SPI_flush();
}

/*********************************
rf_ch_reg_test()*****************************************************
****
 *
 * @name   -  rf_ch_reg_test()
 * @brief  -  function to test read and write operation to nrf module
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

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

/********************************** rf_ch_reg_test() function
definition *********************************/
```

```
void rf_ch_reg_test()
{
	nrf_write_rf_ch(0x79);
	uint8_t channel;
	channel = nrf_read_rf_ch();
	delay();
	transmit(RFCH);
	if(channel==0x79){
		uart_flush("SUCCESSFUL");
	}
	else{
		uart_flush("FAILED");
	}
	SPI_flush();
}

/********************************* status_reg_test()
*************************************************************
 *
 * @name   -  status_reg_test()
 * @brief  -  function to read the status register of nrf module
 * @param  -  none
 *
 * long description - This function reads the status register of nrf
module
 *
 * @return -  void
 *

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

/********************************* status_reg_test() function
definition *********************************/

void status_reg_test()
{
	uint8_t status = nrf_read_status();
	delay();
	transmit(STATUS);
	uart_flush("SUCCESSFUL");
	SPI_flush();
}

/********************************* fifo_status_reg_test()
*************************************************************
 *
 * @name   -  fifo_status_reg_test()
 * @brief  -  function to read the fifo status register of nrf module
 * @param  -  none
 *
```

```
 * long description - This function reads the fifo status register of
nrf module
 *
 * @return -  void
 *

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

/********************************** fifo_status_reg_test() function
definition *********************************/

void fifo_status_reg_test()
{
      uint8_t fifo = nrf_read_fifo_status();
      delay();
      transmit(FIFO);
      uart_flush("SUCCESSFUL");
      SPI_flush();
}

/*********************************
transmit()*************************************************************
 *
 * @name   -  transmit()
 * @brief  -  function to test and log on UART the success of read and
write operations for nrf registers
 * @param  -  result: result value obtained after operations.
 *
 * long description - This function tests and logs on UART the success
of read and write operations for nrf registers
 *
 * @return -  void
 *

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

/********************************** transmit() function definition
**********************************/

void transmit(uint8_t result)
{
      uart_flush("\n\r");
      switch (result){
      case CONFIG:
            uart_flush(CON_REG);
            delay();
            break;
      case TXADDR:
            uart_flush(TXADDR_REG);
            delay();
```

```
            break;
      case RFSETUP:
            uart_flush(RFSETUP_REG);
            delay();
            break;
      case RFCH:
            uart_flush(RFCH_REG);
            delay();
            break;
      case STATUS:
            uart_flush(STAT_REG);
            delay();
            break;
      case FIFO:
            uart_flush(FIFO_REG);
            delay();
            break;
      }
}

/*********************************
lib_memmove_test()*****************************************************
******
 *
 * @name    -  lib_memmove_test()
 * @brief   -  function to test profiling time of memmove (stdlib)
function
 * @param   -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memmove
(stdlib) function
 *
 * @return -  void
 *

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

/********************************** lib_memmove_test() function
definition **********************************/

void lib_memmove_test(uint32_t len)
{
      uint32_t i=0;
      uint8_t source[len];
      while(i<(len))
      {
            source[i]=i;
            i++;
      }
      uint8_t destination[len];
```

```
	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	memmove(source,destination,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

/*********************************
my_memmove_test()*****************************************************
*****
 *
 * @name   -  my_memmove_test()
 * @brief  -  function to test profiling time of memmove (non-dma)
function
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memmove
(non-dma) function.
 *
 * @return -  void
 *

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

/********************************** my_memmove_test() function
definition **********************************/

void my_memmove_test(uint32_t len)
{
	uint32_t i=0;
	uint8_t source[len];
	while(i<(len))
	{
		source[i]=i;
		i++;
	}
```

```
	uint8_t destination[len];

	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	my_memmove(source,destination,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

/*********************************
dma_memmove_test()*****************************************************
******
 *
 * @name   -  dma_memmove_test()
 * @brief  -  function to test profiling time of memmove (dma)
function.
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memmove
(dma) function.
 *
 * @return -  void
 *

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

/********************************** dma_memmove_test() function
definition **********************************/

void dma_memmove_test(uint32_t len)
{
	uint32_t i=0;
	uint8_t source[len];
	while(i<(len))
	{
		source[i]=i;
```

```
		i++;
	}
	uint8_t destination[len];

	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	memmove_dma(source,destination,len,EIGHT_BIT);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

/*********************************
dma_memset_test()*****************************************************
*****
 *
 * @name   -  dma_memset_test()
 * @brief  -  function to test profiling time of memset (dma)
function.
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(dma) function.
 *
 * @return -  void
 *

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

/********************************** dma_memset_test() function
definition **********************************/

void dma_memset_test(uint32_t len)
{
	uint32_t value = 10;
	uint8_t destination[len];
	uint32_t start_time;
```

```
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	memset_dma(value,destination,len,EIGHT_BIT);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

/*********************************
my_memset_test()*****************************************************
****
 *
 * @name   -  my_memset_test()
 * @brief  -  function to test profiling time of memset (non-dma)
function.
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(non-dma) function.
 *
 * @return -  void
 *

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

/********************************** my_memset_test() function
definition **********************************/


void my_memset_test(uint32_t len)
{
	uint32_t value = 10;
	uint8_t destination[len];
	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
```

```
	my_memset(value,destination,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

/*********************************
lib_memset_test()*****************************************************
*****
 *
 * @name   -  lib_memset_test()
 * @brief  -  function to test profiling time of memset (stdlib)
function.
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(stdlib) function.
 *
 * @return -  void
 *

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

/********************************** lib_memset_test() function
definition **********************************/

void lib_memset_test(uint32_t len)
{
	uint32_t value = 10;
	uint8_t destination[len];
	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	memset(destination,value,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);
```

```
	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

void my_memmove_test_optimized(uint32_t len)
{
	uint32_t i=0;
	uint8_t source[len];
	while(i<(len))
	{
		source[i]=i;
		i++;
	}
	uint8_t destination[len];

	uint32_t start_time;
	uint32_t end_time;

	profiler_start();
	start_time=gettime();
	my_memmove_optimized(source,destination,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}

void my_memset_test_optimized(uint32_t len)
{
	uint32_t value = 10;
	uint8_t destination[len];
	uint32_t start_time;
	uint32_t end_time;
```

```
	profiler_start();
	start_time=gettime();
	my_memset_optimized(value,destination,len);
	end_time=gettime();
	profiler_stop();

	t = execution_time(start_time,end_time);

	my_itoa(t,time,10);
	if(Rxd_Data!=44)
	{
		LOG(PROFILING_RESULT,NULL);
	}
	time[0]='\0';
	time[1]='\0';
	time[2]='\0';
	time[3]='\0';
}
```

31) test.h

```
/*************************************************************************
*****************************
 *
 * @author  Preshit Harlikar, Shivam Khandelwal
 * @file test.h
 * @brief This file includes function declarations of functions for
testing profiling, spi and nrf
 * @date December 06, 2017
 *
 * long description - The test.c file includes function declarations
of functions to -
 *                      1) test profile time for memory functions
dma,non-dma,stdlib) (profile_test())
 *                      2) test read and write operation to nrf module
(spi_nrf_test())
 *                      3) test config read and write operation to nrf
module (config_reg_test())
 *                      4) test read and write operation to nrf module
(tx_addr_test())
 *                      5) test read and write operation to nrf module
(rf_setup_test())
 *                      6) test read and write operation to nrf module
(rf_ch_reg_test())
 *                      7) read the status register of nrf module
(status_reg_test())
 *                      8) read the status register of nrf module
(fifo_status_reg_test())
```

```
 *                      9) test and log on UART the success of read
and write operations for nrf registers (transmit())
 *                     10) test profiling time of memmove (stdlib)
function (lib_memmove_test())
 *                     11) test profiling time of memmove (non-dma)
function (my_memmove_test())
 *                     12) test profiling time of memmove (dma)
function (dma_memmove_test())
 *                     13) test profiling time of memset (dma)
function (dma_memset_test())
 *                     14) test profiling time of memset (non-dma)
function (my_memset_test())
 *                     15) test profiling time of memset (stdlib)
function (lib_memset_test())
 *

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

#ifndef SOURCES_TEST_H_
#define SOURCES_TEST_H_

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include "rtc.h"
#include "logger.h"
#include "logbuf.h"
#include "uart.h"
#include "circbuf.h"
#include "gpio.h"
#include "dma.h"
#include "profiler.h"
#include "project3.h"
#include "spi.h"
#include "nordic.h"
#include "memory.h"

enum{
CONFIG=0,
TXADDR=1,
RFSETUP=2,
RFCH=3,
STATUS=4,
FIFO=5
};

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

#define CON_REG            "Config register write     :"
```

```
#define TXADDR_REG          "TXADDR register write     :"
#define RFSETUP_REG   "RFSETUP register write    :"
#define RFCH_REG            "RFCH register write  :"
#define STAT_REG            "STATUS register read :"
#define FIFO_REG      "FIFO register read   :"


/*********************************
profile_test()*************************************************************
**
 *
 * @name   -  profile_test()
 * @brief  -  function to test profile time for memory functions
dma,non-dma,stdlib)
 * @param  -  none
 *
 * long description - This function passes log struct to log buffer.
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void profile_test();

/*********************************
spi_nrf_test()()***********************************************************
****
 *
 * @name   -  spi_nrf_test()
 * @brief  -  function to test read and write operation to nrf module
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void spi_nrf_test();

/*********************************
dma_memmove_test()*********************************************************
******
 *
 * @name   -  dma_memmove_test()
 * @brief  -  function to test profiling time of memmove (dma)
function.
 * @param  -  len: length of bytes to be tested
 *
```

```
 * long description - This function tests profiling time of memmove
(dma) function.
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void dma_memmove_test(uint32_t len);

/*********************************
dma_memset_test()*****************************************************
*****
 *
 * @name   -  dma_memset_test()
 * @brief  -  function to test profiling time of memset (dma)
function.
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(dma) function.
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void dma_memset_test(uint32_t len);

/*********************************
my_memmove_test()*****************************************************
*****
 *
 * @name   -  my_memmove_test()
 * @brief  -  function to test profiling time of memmove (non-dma)
function
 * @param  -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memmove
(non-dma) function.
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void my_memmove_test(uint32_t len);

/*********************************
my_memset_test()******************************************************
****
 *
```

```
 * @name    -  my_memset_test()
 * @brief   -  function to test profiling time of memset (non-dma)
function.
 * @param   -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(non-dma) function.
 *
 * @return -  void
 *

*************************************************************************
*******************************/
void my_memset_test(uint32_t len);

void my_memmove_test_optimized(uint32_t len);

void my_memset_test_optimized(uint32_t len);


/*********************************
lib_memmove_test()*****************************************************
******
 *
 * @name    -  lib_memmove_test()
 * @brief   -  function to test profiling time of memmove (stdlib)
function
 * @param   -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memmove
(stdlib) function
 *
 * @return -  void
 *

*************************************************************************
*******************************/
void lib_memmove_test(uint32_t len);

/*********************************
lib_memset_test()******************************************************
*****
 *
 * @name    -  lib_memset_test()
 * @brief   -  function to test profiling time of memset (stdlib)
function.
 * @param   -  len: length of bytes to be tested
 *
 * long description - This function tests profiling time of memset
(stdlib) function.
 *
 * @return -  void
```

```
 *
**************************************************************************
*******************************/
void lib_memset_test(uint32_t len);

/*********************************
config_reg_test()*****************************************************
*****
 *
 * @name   -  config_reg_test()
 * @brief  -  function to test config read and write operation to nrf
module.
 * @param  -  none
 * long description - This function tests config read and write
operation to nrf module.
 *
 * @return -  void
 *

**************************************************************************
*******************************/
void config_reg_test();

/*********************************
tx_addr_test()********************************************************
**
 *
 * @name   -  tx_addr_test()
 * @brief  -  function to test read and write operation to nrf module.
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

**************************************************************************
*******************************/
void tx_addr_test();

/*********************************
rf_setup_test()*******************************************************
***
 *
 * @name   -  rf_setup_test()
 * @brief  -  function to test read and write operation to nrf module.
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module.
```

```
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void rf_setup_test();

/*********************************
rf_ch_reg_test()*******************************************************
****
 *
 * @name   -  rf_ch_reg_test()
 * @brief  -  function to test read and write operation to nrf module
 * @param  -  none
 *
 * long description - This function tests read and write operation to
nrf module
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void rf_ch_reg_test();

/********************************** status_reg_test()
*********************************************************
 *
 * @name   -  status_reg_test()
 * @brief  -  function to read the status register of nrf module
 * @param  -  none
 *
 * long description - This function reads the status register of nrf
module
 *
 * @return -  void
 *

**********************************************************************
*******************************/
void status_reg_test();

/********************************** fifo_status_reg_test()
*********************************************************
 *
 * @name   -  fifo_status_reg_test()
 * @brief  -  function to read the fifo status register of nrf module
 * @param  -  none
 *
 * long description - This function reads the fifo status register of
nrf module
```

```
 *
 * @return -  void
 *

*****************************************************************************
*******************************/
void fifo_status_reg_test();

/*********************************
transmit()*****************************************************************
 *
 * @name   -  transmit()
 * @brief  -  function to test and log on UART the success of read and
write operations for nrf registers
 * @param  -  result: result value obtained after operations.
 *
 * long description - This function tests and logs on UART the success
of read and write operations for nrf registers
 *
 * @return -  void
 *

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

void transmit(uint8_t result);

#endif /* SOURCES_TEST_H_ */
```