```
/**
 * @\file   uart.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for
uart transmit and receive on BBG
 * @\date   04/29/2019
 * References : https://github.com/sijpesteijn/BBCLib,
 *
      https://en.wikibooks.org/wiki/Serial_Programming/termios
 *
 */

#ifndef UART_H
#define UART_H

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <stdlib.h>
#include <stdint.h>

#include "logger.h"
#include "heartbeat.h"

/* Structure to store sensor data and mode of operation to send to remote
request task */
typedef struct {
      float lux;
      float distance;
      float waterLevel;
      int8_t mode;
      int8_t dg_mode;
} communication;

communication comm_rec;

typedef enum {
      uart00 = 0, uart01 = 1, uart02 = 2, uart03 = 3, uart04 = 4, uart05
= 5
} uart;


/* Structure to initialize particular UART on BBG */
typedef struct {
      int fd;
      uart uart_no;
      int baudrate;
}uart_properties;

uart_properties  *uart2;
```

```
/**
------------------------------------------------------------------------
-------------------
uart_config
------------------------------------------------------------------------
-------------------
*   This function will configure the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                       success
*                           -1                          failure
*
*/
int8_t uart_config(uart_properties *uart);


/**
------------------------------------------------------------------------
-------------------
uart_close
------------------------------------------------------------------------
-------------------
*   This function will close the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                       success
*
*/
int8_t uart_close(uart_properties *uart);


/**
------------------------------------------------------------------------
-------------------
uart_send
------------------------------------------------------------------------
-------------------
*   This function will send specified length of bytes over the UART on
BBG
*
*   @\param         uart_properties     specifies UART number
*                           tx                          byte to be sent
*                           length                      number of bytes
to be sent
*
*   @\return        -1                      Failure
*                            count                      Number of bytes
sent
*
*/
int8_t uart_send(uart_properties *uart, void *tx, int length);
```

```
/**
------------------------------------------------------------------------
-------------------
uart_receive
------------------------------------------------------------------------
-------------------
*   This function will receive tiva sensor data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                           rx_r                    byte received
*                           length                      number of bytes
to be received
*
*   @\return        -1                      Failure
*                            count                      Number of bytes
sent
*
*/
int8_t uart_receive(uart_properties *uart, void *rx_r, int length);


/**
------------------------------------------------------------------------
-------------------
uart_receive_task
------------------------------------------------------------------------
-------------------
*   This function will receive tiva log data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                           rx_r                    byte received
*                           length                      number of bytes
to be received
*
*   @\return        -1                      Failure
*                            count                      Number of bytes
sent
*
*/
int8_t uart_receive_task(uart_properties *uart, void *rx_r, int length);
#endif
/**
 * @\file   logger.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
logger
 * @\date   04/29/2019
 *
 */

#ifndef LOGGER_H
#define LOGGER_H
```

```
#include "uart.h"
#include <pthread.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h> // mkdir
#include <mqueue.h>

#define QUEUE_NAME "/msg_queue" // Message queue name
#define MAX_BUFFER_SIZE      200     // Message queue max buffer size


/* Sensor structure that is sent from Tiva */
typedef struct sensor_struct
{
    char task_name[5];
    uint32_t timeStamp;

    float distance;
    float lux;
    uint32_t water;
    int8_t mode;
    int8_t dg_mode;
}sensor_struct;

/* Logger structure that is sent from Tiva */
typedef struct logger_struct
{
      char task_name[5];
      uint32_t timeStamp;

      char log[100];
}logger_struct;

pthread_t logger_thread;

FILE *file_ptr;

mqd_t msg_queue; // Message queue descriptor

extern pthread_mutex_t lock_res; // Mutex


/**
------------------------------------------------------------------------
-------------------
time_stamp
------------------------------------------------------------------------
-------------------
*   This function will format the timestamp
*
*   @\param
*
*   @\return        timestamp as a string
*
```

```
*/
char *time_stamp();


/**
------------------------------------------------------------------------
-------------------
logger_thread_callback
------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void *logger_thread_callback();


/**
------------------------------------------------------------------------
-------------------
logger_init
------------------------------------------------------------------------
-------------------
*   This function will initialize the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void logger_init();
#endif


/**
 * @\file   communication.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
communication interface for tiva and BBG
 * @\date   04/29/2019
 *
 */
#ifndef COMMUNICATION_H
#define COMMUNICATION_H

#include <pthread.h>
#include <stdint.h>

#include "uart.h"
#include "logger.h"
#include "POSIX_timer.h"
```

```
#include "heartbeat.h"


/*
*     GLOBALS
*/
char lux[10];
char distance[10];
char waterLevel[10];
char mode[10];
char dg_mode[10];
char tiva_opstatus[10];
char distance_opstatus[10];
char lux_opstatus[10];
char water_opstatus[10];
char valve_status[10];

pthread_t communication_thread;

uint8_t already_open;
uint8_t already_closed;
uint8_t water_outOfRange;

extern pthread_mutex_t lock_res;


/*
------------------------------------------------------------------------
-------------------
communication_thread_callback
------------------------------------------------------------------------
-------------------
*     This is the thread creation callback function for the communication
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *communication_thread_callback();


/*
------------------------------------------------------------------------
-------------------
get_lux
------------------------------------------------------------------------
-------------------
*     This function returns the most updated lux value in string format
*
*     @\param                void
*
*     @\return         string containing most recent lux value
```

```
*
*/
char *get_lux();


/*
-------------------------------------------------------------------------
-------------------
get_distance
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated distance value in string
format
*
*     @\param               void
*
*     @\return         string containing most recent distance value
*
*/
char *get_distance();


/*
-------------------------------------------------------------------------
-------------------
get_waterLevel
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated water level value in string
format
*
*     @\param               void
*
*     @\return         string containing most recent water level value
*
*/
char *get_waterLevel();


/*
-------------------------------------------------------------------------
-------------------
get_mode
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operating mode in string
format
*
*     @\param               void
*
*     @\return         string containing most updated operating mode
value
*
*/
```

```
char *get_mode();


/*
-------------------------------------------------------------------------
-------------------
get_dgMode
-------------------------------------------------------------------------
-------------------
*     This function returns if the system is in degraded mode in string
format
*
*     @\param               void
*
*     @\return         string containing degraded mode value
*
*/
char *get_dgMode();


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_tiva
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
remote node
*
*     @\param               void
*
*     @\return         string containing the most updated operation
status of the remote node
*
*/
char *get_opStatus_tiva();


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_distance
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
distance sensor
*
*     @\param               void
*
*     @\return         string containing the most updated operation
status of the distance sensor
*
*/
char *get_opStatus_distance();
```

```
/*
-------------------------------------------------------------------------
-------------------
get_opStatus_lux
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the lux
sensor
*
*     @\param               void
*
*     @\return         string containing the most updated operation
status of the lux sensor
*
*/
char *get_opStatus_lux();


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_water
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
water level sensor
*
*     @\param               void
*
*     @\return         string containing the most updated operation
status of the water level sensor
*
*/
char *get_opStatus_water();


/*
-------------------------------------------------------------------------
-------------------
get_valveStatus
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated status of the valve
*
*     @\param               void
*
*     @\return         string containing the most updated status of the
valve
*
*/
char *get_valveStatus();
#endif
```

```
/**
 * @\file   server.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
server socket
 * @\date   04/29/2019
 *
 */

#ifndef SERVER_H
#define SERVER_H

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <errno.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <stdbool.h>

#include "communication.h"

#define PORT_NO 8005

extern pthread_mutex_t lock_res;

pthread_t remote_request_thread;
/*********************************************
       Function for server socket creation
       Parameters : Port number
*********************************************/
int socket_creation_server(int port);


/************************************************
       Function for remote request thread creation
       Parameters :
************************************************/
void *remote_request_callback();

#endif
/**
 * @\file   heartbeat.h
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the declarations and header files for the
heartbeat
 * @\date   04/29/2019
 *
 */
```

```
#ifndef HEARTBEAT_H
#define HEARTBEAT_H

#include <pthread.h>
#include "POSIX_timer.h"

#include "communication.h"
#include "uart.h"
#include "logger.h"

#define TIVA_HEART_BEAT_CHECK_PERIOD (5) //5 seconds


/* Global variables for dead/alive status detection of Tiva and sensors
*/
int tiva_active, tiva_active_prev;
int distance_active, distance_active_prev, lux_active, lux_active_prev,
water_active, water_active_prev;

int tiva_dead, distance_dead, lux_dead, water_dead;


timer_t  timer_id_heartbeat;

pthread_t heartbeat_thread;

extern pthread_mutex_t lock_res; // Mutex


/*
------------------------------------------------------------------------
--------------------
heartbeat_thread_callback
------------------------------------------------------------------------
--------------------
*     This is the thread creation callback function for the heartbeat
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *heartbeat_thread_callback();


/*
------------------------------------------------------------------------
--------------------
beat_timer_handler
------------------------------------------------------------------------
--------------------
*     This function is the timer handler for tiva heart beat timer
*
```

```
*     @\param                signal value ( dummy)
*
*     @\return         none
*
*/
void beat_timer_handler(union sigval val);

#endif
/**
 * @\file   log_receiver.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
tiva log receiver module
 * @\date   03/30/2019
 *
 */


#ifndef LOG_RECEIVER_H
#define LOG_RECEIVER_H

#include "uart.h"
#include "logger.h"

pthread_t log_receiver_thread;


/**
------------------------------------------------------------------------------
-------------------
revecive_thread_callback
------------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for logger messages
coming from Tiva
*
*   @\param         void
*
*   @\return        void
*
*/
void *revecive_thread_callback();

#endif/**
 * @\file   POSIX_timer.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the declarations and header files for
POSIX timer modules
 * @\date   04/29/2019
 *
 */

#ifndef POSIX_Timer_H_
#define POSIX_Timer_H_
```

```
/*****************************************************************
                                    Includes
*****************************************************************/
#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <time.h>
#include <stdlib.h>
#include <signal.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/stat.h>
#include <errno.h>

enum Status{SUCCESS = 0, ERROR = -1, LUX_ERROR = -2, REMOTE_SOCKET_ERROR
= -3, LOGGER_ERROR = -4,  TEMP_ERROR = -1000, BRIGHT = 1000, DARK = -
1000};

/*****************************************************************
                                    Function Protypes
*****************************************************************/
/*
------------------------------------------------------------------------
-------------------
kick_timer
------------------------------------------------------------------------
-------------------
*     This helps in restarting the timer after expiration
*
*     @\param                timer descriptor, timer expiration time in
ns
*
*     @\return         error status
*
*/
int kick_timer(timer_t, int);

/*
------------------------------------------------------------------------
-------------------
setup_timer_POSIX
------------------------------------------------------------------------
-------------------
*     This helps in creating the timer
*
*     @\param                timer descriptor, timer handler function
*
*     @\return         error status
*
*/
```

```
int setup_timer_POSIX(timer_t *,void (*handler)(union sigval));

/*
------------------------------------------------------------------------------
-------------------
stop_timer
------------------------------------------------------------------------------
-------------------
*     This helps in deleting the timer
*
*     @\param                timer descriptor
*
*     @\return          error status
*
*/
int stop_timer(timer_t);

/*
------------------------------------------------------------------------------
-------------------
temp_timer_handler
------------------------------------------------------------------------------
-------------------
*     This is the timer handler for temperature timer
*
*     @\param                dummy
*
*     @\return          error status
*
*/
void temp_timer_handler(union sigval);

/*
------------------------------------------------------------------------------
-------------------
lux_timer_handler
------------------------------------------------------------------------------
-------------------
*     This is the timer handler for lux timer
*
*     @\param                dummy
*
*     @\return          error status
*
*/
void lux_timer_handler(union sigval);

/*
------------------------------------------------------------------------------
-------------------
log_timer_handler
------------------------------------------------------------------------------
-------------------
*     This is the timer handler for log timer
```

```
*
*     @\param                dummy
*
*     @\return         error status
*
*/
void log_timer_handler(union sigval);

/*****************************************************************
                                    MACROS
*****************************************************************/
#define Delay_NS (2000000000)//2000ms


#endif /* POSIX_Timer_H_ */
/**
 * @\file   POSIX_timer.h
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for POSIX timer
 * @\date   04/29/2019
 *
 */

/*****************************************************************
                                    Includes
*****************************************************************/
#include "POSIX_timer.h"



/*****************************************************************
                              POSIX Timer configuration
*****************************************************************/
int setup_timer_POSIX(timer_t *timer_id,void (*handler)(union sigval))
{
      struct      sigevent sev;
      sev.sigev_notify = SIGEV_THREAD; //Upon timer expiration, invoke
sigev_notify_function
      sev.sigev_notify_function = handler; //this function will be called
when timer expires
      sev.sigev_notify_attributes = NULL;
      sev.sigev_value.sival_ptr = &timer_id;


      if(timer_create(CLOCK_REALTIME, &sev, timer_id) != 0) //on success,
timer id is placed in timer_id
      {
            return ERROR;
      }



    return SUCCESS;
```

```
}

/*****************************************************************
                            Start configuration
                Parameter : delay in nano secs
*****************************************************************/
int kick_timer(timer_t timer_id, int interval_s)
{
   struct itimerspec in;

    in.it_value.tv_sec = interval_s; //sets initial time period
    in.it_value.tv_nsec = 0;
    in.it_interval.tv_sec = interval_s; //sets interval
    in.it_interval.tv_nsec =0;
    //issue the periodic timer request here.
    if( (timer_settime(timer_id, 0, &in, NULL)) != SUCCESS)
    {
      return ERROR;
    }
    return SUCCESS;
}

/*****************************************************************
                            Destroy Timer
*****************************************************************/
int stop_timer(timer_t timer_id)
{
      if( (timer_delete(timer_id)) != SUCCESS)
    {
      printf("Error on delete timer function\n");
      return ERROR;
    }


    return SUCCESS;
}
/**
 * @\file   server.c
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for server
socket
 * @\date   04/29/2019
 *
 */

#include "server.h"
/*****************************************************************
                        Globals
*****************************************************************/
socklen_t clilen;
struct sockaddr_in to_address;

int new_socket, server_socket;
/*********************************************
```

```
       Function for server socket creation
       Parameters : Port number
*******************************************/
int socket_creation_server(int port)
{

        //creating the socket for client

        server_socket = socket(AF_INET,SOCK_STREAM,0);// setting the
client socket
        if(server_socket < 0 ) // enters this loop if port number is not
given as command line argument
        {
                //printing error message when opening client socket
          perror("Error opening server socket\n");
          return -1;
        }


        struct sockaddr_in server_address;

        memset(&server_address,0,sizeof(server_address));

        //assigning values for the server address structure
        server_address.sin_family = AF_INET;
        server_address.sin_port = htons(port); // converting to network
byte order
        server_address.sin_addr.s_addr = INADDR_ANY;



        if(bind(server_socket,(struct
sockaddr*)&server_address,sizeof(server_address))<0)
        {
          perror("Binding failed in the server");
          return -1;
        }

        /*Listening for clients*/
        if(listen(server_socket,10) < 0)
        {
          perror("Error on Listening ");
          return -1;
        }
        else
        {
          printf("\nlistening to remote requests.....\n");
        }

      return 0;

}

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

```
       Function for remote request thread creation
       Parameters : Structure typecasted to void *
************************************************/
void *remote_request_callback()
{


    char buffer[10];
    char manual = 'm';
    char automatic = 'a';
    char up = 'u';
    char down = 'b';
    char left = 'l';
    char right = 'r';
    char stop = 's';
    char clean = 'o';

    char mode_send[10];
    char waterLevel_send[10];
    char distance_send[10];
    char lux_send[10];
    char dgMode_send[10];
    char tiva_opstatus_send[10];
    char distance_opstatus_send[10];
    char lux_opstatus_send[10];
    char water_opstatus_send[10];
    char valve_status_send[10];

//creating socket for server
    if(socket_creation_server(PORT_NO)== -1)
    {
      perror("Error on socket creation - killed remote request socket");

    }


    while(1)
    {
      new_socket = 0;
      memset(&to_address,0,sizeof(to_address));


      clilen = sizeof(to_address);

    /*accepting client requests*/
      new_socket = accept(server_socket,(struct sockaddr*) &to_address,
&clilen);
      if(new_socket<0)
      {
        perror("Error on accepting client");
      }
      else
      {
        printf("established connection\n");
```

```
      }

    /*Forked the request received so as to accept multiple clients*/
      int child_id = 0;
    /*Creating child processes*/
    /*Returns zero to child process if there is successful child
creation*/
      child_id = fork();

    // error on child process
      if(child_id < 0)
      {
        perror("error on creating child\n");
        exit(1);
      }

    //closing the parent
      if (child_id > 0)
      {
        close(new_socket);
        waitpid(0, NULL, WNOHANG);  //Wait for state change of the child
process
      }

      if(child_id == 0)
      {

        memset(buffer,'\0',sizeof(buffer));
        while(recv(new_socket, buffer ,10, 0) > 0)
        {


         // printf("Received request %s - %ld\n",buffer,strlen(buffer));

         if(strcmp(buffer,"display")==0)
         {
          //printf("Received request for display\n");

          strcpy(lux_send, get_lux());
          send(new_socket, lux, 10 , 0);

          strcpy(distance_send, get_distance());
          send(new_socket, distance_send, 10 , 0);

          strcpy(waterLevel_send, get_waterLevel());
          send(new_socket, waterLevel_send, 10, 0);

              strcpy(mode_send, get_mode());
              send(new_socket, mode_send, 10, 0);

              strcpy(dgMode_send, get_dgMode());
              send(new_socket, dgMode_send, 10, 0);

          strcpy(tiva_opstatus_send, get_opStatus_tiva());
```

```
  send(new_socket, tiva_opstatus_send, 10, 0);

  strcpy(distance_opstatus_send, get_opStatus_distance());
  send(new_socket, distance_opstatus_send, 10, 0);

  strcpy(lux_opstatus_send, get_opStatus_lux());
  send(new_socket, lux_opstatus_send, 10, 0);

  strcpy(water_opstatus_send, get_opStatus_water());
  send(new_socket, water_opstatus_send, 10, 0);

  strcpy(valve_status_send, get_valveStatus());
  send(new_socket, valve_status_send, 10, 0);

  }

else if(strcmp(buffer,"manual")==0)
{
  //send m to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &manual, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"auto")==0)
{
      printf("AUTO\n");
        //send a to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &automatic, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"up")==0)
{
        //send u to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &up, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"down")==0)
{
        //send b to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &down, sizeof(char));
  pthread_mutex_unlock(&lock_res);
}

else if(strcmp(buffer,"left")==0)
{
        //send l to tiva
  pthread_mutex_lock(&lock_res);
  uart_send(uart2, &left, sizeof(char));
```

```
          pthread_mutex_unlock(&lock_res);
        }

        else if(strcmp(buffer,"right")==0)
        {
                //send r to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &right, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

        else if(strcmp(buffer,"stop")==0)
        {
                //send s to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &stop, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

        if(strcmp(buffer, "on") == 0)
        {
          printf("ON RECEIVED\n");
          //send o to tiva
          pthread_mutex_lock(&lock_res);
          uart_send(uart2, &clean, sizeof(char));
          pthread_mutex_unlock(&lock_res);
        }

      }
      close(new_socket);
      exit(0);
    }
  }
}

#include <stdio.h>
#include <pthread.h>
#include <string.h>
#include <sys/types.h>

#include "server.h"
#include "communication.h"
#include "logger.h"
#include "heartbeat.h"
#include "log_receiver.h"


pthread_mutex_t lock_res; // Mutex

int main()
{
      char buffer[MAX_BUFFER_SIZE];

      pthread_attr_t attr;
```

```
      pthread_attr_init(&attr);

      logger_init();

      /* Create logger thread */
      if(pthread_create(&logger_thread, &attr, logger_thread_callback,
NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"ERROR CN [%s] Logger thread creation
failed\n",time_stamp());
           mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Logger thread creation failed");
      }

      if (pthread_mutex_init(&lock_res, NULL) != 0)
    {
        perror("Mutex init failed\n");
        memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Mutex init
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
        return -1;
    }

      if(pthread_create(&remote_request_thread, &attr,
remote_request_callback, NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"ERROR CN [%s] Remote request thread creation
failed\n",time_stamp());
           mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Remote socket thread creation failed");
      }

      if(pthread_create(&communication_thread, &attr,
communication_thread_callback, NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"ERROR CN [%s] Communication thread creation
failed\n",time_stamp());
           mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Communication thread creation failed");
      }

      if(pthread_create(&log_receiver_thread, &attr,
revecive_thread_callback, NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"ERROR CN [%s] Logger receive thread creation
failed\n",time_stamp());
           mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Receiver logger thread creation failed");
      }
```

```
      if(pthread_create(&heartbeat_thread, &attr,
heartbeat_thread_callback, NULL) != 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Heartbeat thread creation
failed\n",time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Heartbeat thread creation failed");
      }

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Threads creation
success\n",time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      printf("Threads created successfully\n");

      pthread_join(communication_thread,NULL);
      pthread_join(logger_thread,NULL);
      pthread_join(log_receiver_thread,NULL);
      pthread_join(heartbeat_thread, NULL);
      pthread_join(remote_request_thread,NULL);

      return 0;
}
/**
 * @\file   heartbeat.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the definitions for the heartbeat
functions
 * @\date   04/29/2019
 *
 */

#include "heartbeat.h"


/*
------------------------------------------------------------------------
-------------------
beat_timer_handler
------------------------------------------------------------------------
-------------------
*     This function is the timer handler for tiva heart beat timer
*
*     @\param                signal value ( dummy)
*
*     @\return         none
*
*/
void beat_timer_handler(union sigval val)
{
      char buffer[MAX_BUFFER_SIZE];
```

```
if(tiva_active <= tiva_active_prev)
{
      tiva_dead = 1;
      printf("ERROR Tiva dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Tiva Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      tiva_dead = 0;
      printf("INFO Tiva alive\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"DEBUG CN [%s] Tiva Alive\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}

if(distance_active <= distance_active_prev)
{
      distance_dead = 1;
      printf("ERROR Ultrasonic dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Ultrasonic Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      distance_dead = 0;

      printf("INFO Ultrasonic alive\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"DEBUG CN [%s] Ultrasonic Alive\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}

if(lux_active <= lux_active_prev)
{
      lux_dead = 1;
      printf("ERROR Lux dead\n");

      memset(buffer,'\0',sizeof(buffer));
sprintf(buffer,"ERROR CN [%s] Lux Dead\n", time_stamp());
mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
}
else
{
      lux_dead = 0;

      printf("INFO Lux alive\n");
```

```
            memset(buffer,'\0',sizeof(buffer));
        sprintf(buffer,"DEBUG CN [%s] Lux Alive\n", time_stamp());
        mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
        }

        if((water_active <= water_active_prev) || water_outOfRange)
        {
            water_dead = 1;
            printf("ERROR Water dead\n");

            memset(buffer,'\0',sizeof(buffer));
        sprintf(buffer,"ERROR CN [%s] Water level Dead\n", time_stamp());
        mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
        }
        else
        {
            water_dead = 0;

            printf("INFO Water alive\n");

            memset(buffer,'\0',sizeof(buffer));
        sprintf(buffer,"DEBUG CN [%s] Water level Alive\n", time_stamp());
        mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
        }


        tiva_active_prev = tiva_active;
        distance_active_prev = distance_active;
        lux_active_prev = lux_active;
        water_active_prev = water_active;



        //restarting the heartbeat timer
        if((kick_timer(timer_id_heartbeat, TIVA_HEART_BEAT_CHECK_PERIOD))
== -1)
        {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"ERROR CN [%s] Kicking timer for heartbeat
failed\n", time_stamp());
           mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Error on kicking timer for heartbeat\n");
        }

}

/*
------------------------------------------------------------------------
-------------------
heartbeat_thread_callback
------------------------------------------------------------------------
-------------------
```

```
*     This is the thread creation callback function for the heartbeat
thread
*
*     @\param                void
*
*     @\return         void
*
*/
void *heartbeat_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Heartbeat thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      printf("Inside heartbeat thread\n");


      if((setup_timer_POSIX(&timer_id_heartbeat,beat_timer_handler)) == -
1)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Creating timer for heartbeat
failed\n", time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Error on creating timer for heartbeat\n");
      }

      if((kick_timer(timer_id_heartbeat, TIVA_HEART_BEAT_CHECK_PERIOD))
== -1)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"ERROR CN [%s] Kicking timer for heartbeat
failed\n", time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
            perror("Error on kicking timer for heartbeat\n");
      }

      /* Configure UART2 on BBG */
      uart2  = malloc(sizeof(uart_properties));
      uart2->uart_no = 2;
      uart2->baudrate = B115200;

      uint8_t isOpen2 = uart_config(uart2);

      printf("IS OPEN 2: %d\n",isOpen2);
      char hb ='h';

      if(isOpen2 == 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"DEBUG CN [%s] UART2 Opened successfully\n",
time_stamp());
```

```
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  usleep(1000000); // Sending heartbeat every 1 second

                  pthread_mutex_lock(&lock_res);
                  uart_send(uart2,&hb,sizeof(char));
                  pthread_mutex_unlock(&lock_res);

            }
      }
      uart_close(uart2); // Close UART2
      pthread_cancel(heartbeat_thread);
      return 0;

}
/**
 * @\file   logger.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the function definitions for the logger
 * @\date   04/29/2019
 *
 */

#include "logger.h"

pthread_mutex_t lock; // Mutex

char time_stam[30];

/**
--------------------------------------------------------------------------
-------------------
time_stamp
--------------------------------------------------------------------------
-------------------
*   This function will format the timestamp
*
*   @\param
*
*   @\return        timestamp as a string
*
*/
char *time_stamp()
{

    memset(time_stam,'\0',30);
      time_t timer;
      timer = time(NULL);
      strftime(time_stam, 26, "%Y-%m-%d %H:%M:%S", localtime(&timer));
      return time_stam;
}
```

```
/**
------------------------------------------------------------------------
-------------------
logger_init
------------------------------------------------------------------------
-------------------
*   This function will initialize the logger
*
*   @\param         void
*
*   @\return        void
*
*/
void logger_init()
{
      mq_unlink(QUEUE_NAME);
      file_ptr = fopen("test.log", "w");
      fprintf(file_ptr,"Queue Init\n\n");
      fclose(file_ptr);

    if (pthread_mutex_init(&lock, NULL) != 0)
    {
        perror("Mutex init failed\n");
    }


    struct mq_attr mq_attributes;

    /* Setting the message queue attributes */
    mq_attributes.mq_flags = 0;
    mq_attributes.mq_maxmsg = 5;
    mq_attributes.mq_msgsize = MAX_BUFFER_SIZE;
    mq_attributes.mq_curmsgs = 0;

    msg_queue = mq_open(QUEUE_NAME, O_CREAT | O_RDWR | O_NONBLOCK, 0666,
&mq_attributes);
    perror("MQ FAILED");
    printf("Return value of queue open = %d\n", msg_queue);
}


/**
------------------------------------------------------------------------
-------------------
logger_thread_callback
------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for the logger
*
*   @\param         void
*
*   @\return        void
*
```

```
*/
void *logger_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];

      printf("Inside logger thread\n");

      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Logger thread active\n", time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      file_ptr = fopen("test.log", "a");
      while(1)
      {

            if(mq_receive(msg_queue, buffer,MAX_BUFFER_SIZE,0) > 0)
            {
                  pthread_mutex_lock(&lock);
                  fprintf(file_ptr,"%s",buffer);
                  fflush(file_ptr);
                  pthread_mutex_unlock(&lock);
            }
      }
      fclose(file_ptr);
      mq_close(msg_queue);
    mq_unlink(QUEUE_NAME);
    pthread_cancel(logger_thread);
}


/**
 * @\file   communication.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the definitions for the communication
interface for tiva and BBG
 * @\date   04/29/2019
 *
 */
#include "communication.h"

/*
-------------------------------------------------------------------------
-------------------
get_lux
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated lux value in string format
*
*     @\param                void
*
*     @\return          string containing most recent lux value
*
*/
char *get_lux()
```

```
{
      memset(lux,'\0',sizeof(lux));
      sprintf(lux,"%f",comm_rec.lux);
      return lux;
}


/*
------------------------------------------------------------------------
-------------------
get_distance
------------------------------------------------------------------------
-------------------
*     This function returns the most updated distance value in string
format
*
*     @\param                void
*
*     @\return          string containing most recent distance value
*
*/
char *get_distance()
{
      memset(distance,'\0',sizeof(distance));
      sprintf(distance,"%f",comm_rec.distance);
      return distance;
}


/*
------------------------------------------------------------------------
-------------------
get_waterLevel
------------------------------------------------------------------------
-------------------
*     This function returns the most updated water level value in string
format
*
*     @\param                void
*
*     @\return          string containing most recent water level value
*
*/
char *get_waterLevel()
{
      memset(waterLevel,'\0', sizeof(waterLevel));
      sprintf(waterLevel,"%f", comm_rec.waterLevel);
      return waterLevel;
}


/*
------------------------------------------------------------------------
-------------------
```

```
get_mode
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated operating mode in string
format
*
*     @\param                void
*
*     @\return          string containing most updated operating mode
value
*
*/
char *get_mode()
{
      memset(mode,'\0', sizeof(mode));
      if(!comm_rec.mode)
            sprintf(mode,"Auto");
      else if(comm_rec.mode)
            sprintf(mode,"Manual");
      return mode;
}

/*
--------------------------------------------------------------------------
-------------------
get_dgMode
--------------------------------------------------------------------------
-------------------
*     This function returns if the system is in degraded mode in string
format
*
*     @\param                void
*
*     @\return          string containing degraded mode value
*
*/
char *get_dgMode()
{
      memset(dg_mode,'\0', sizeof(dg_mode));
      if(!comm_rec.dg_mode)
            sprintf(dg_mode,"Normal");
      else if(comm_rec.dg_mode)
            sprintf(dg_mode,"Degraded");

      return dg_mode;
}

/*
--------------------------------------------------------------------------
-------------------
get_opStatus_tiva
--------------------------------------------------------------------------
-------------------
```

```
*     This function returns the most updated operation status of the
remote node
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the remote node
*
*/
char *get_opStatus_tiva()
{
      memset(tiva_opstatus,'\0', sizeof(tiva_opstatus));
      if(!tiva_dead)
            sprintf(tiva_opstatus,"Alive");
      if(tiva_dead)
            sprintf(tiva_opstatus,"Dead");

      return tiva_opstatus;
}


/*
------------------------------------------------------------------------
--------------------
get_opStatus_distance
------------------------------------------------------------------------
--------------------
*     This function returns the most updated operation status of the
distance sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the distance sensor
*
*/
char *get_opStatus_distance()
{
      memset(distance_opstatus,'\0', sizeof(distance_opstatus));
      if(!distance_dead)
            sprintf(distance_opstatus,"Alive");
      if(distance_dead)
            sprintf(distance_opstatus,"Dead");

      return distance_opstatus;
}


/*
------------------------------------------------------------------------
--------------------
get_opStatus_lux
------------------------------------------------------------------------
--------------------
```

```
*     This function returns the most updated operation status of the lux
sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the lux sensor
*
*/
char *get_opStatus_lux()
{

      memset(lux_opstatus,'\0', sizeof(lux_opstatus));
      if(!lux_dead)
            sprintf(lux_opstatus,"Alive");
      if(lux_dead)
            sprintf(lux_opstatus,"Dead");

      return lux_opstatus;
}


/*
-------------------------------------------------------------------------
-------------------
get_opStatus_water
-------------------------------------------------------------------------
-------------------
*     This function returns the most updated operation status of the
water level sensor
*
*     @\param                void
*
*     @\return         string containing the most updated operation
status of the water level sensor
*
*/
char *get_opStatus_water()
{
      memset(water_opstatus,'\0', sizeof(water_opstatus));
      if(water_dead || water_outOfRange)
            sprintf(water_opstatus,"Dead");
      else
            sprintf(water_opstatus,"Alive");


      return water_opstatus;
}


/*
-------------------------------------------------------------------------
-------------------
get_valveStatus
```

```
--------------------------------------------------------------------------
-------------------
*     This function returns the most updated status of the valve
*
*     @\param               void
*
*     @\return         string containing the most updated status of the
valve
*
*/
char *get_valveStatus()
{
      memset(water_opstatus,'\0', sizeof(water_opstatus));
      if(already_open)
            sprintf(valve_status,"Open");
      else if(already_closed)
            sprintf(valve_status,"Closed");

      return valve_status;

}



/*
--------------------------------------------------------------------------
-------------------
communication_thread_callback
--------------------------------------------------------------------------
-------------------
*     This is the thread creation callback function for the communication
thread
*
*     @\param               void
*
*     @\return         void
*
*/
void *communication_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];
      /* Flags to check tiva and sensor heartbeats */
      tiva_active = 0;

      tiva_active_prev = 0;

      distance_active_prev = 0;

      distance_active = 0;

      lux_active_prev = 0;

      lux_active = 0;
```

```
      water_active_prev = 0;

    water_active = 0;

    memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Communication thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);
      printf("Inside communication thread\n");

      struct sensor_struct sensor;

      /* Configure UART4 on BBG */
      uart_properties *uart4 = malloc(sizeof(uart_properties));
      uart4->uart_no = 4;
      uart4->baudrate = B115200;

      uint8_t isOpen4 = uart_config(uart4);

      printf("Open success? %d\n", isOpen4);

      /* Messages to be sent to Tiva for closed loop control */
      char obj_detect = '1';
      char valve_close = '2';
      char valve_open = '3';
      char lux_auto = '4';

      int recv = 0;


      if (isOpen4 == 0) {
            memset(buffer,'\0',sizeof(buffer));
           sprintf(buffer,"DEBUG CN [%s] UART4 Opened successfully\n",
time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  recv = uart_receive(uart4,&sensor,
sizeof(sensor_struct)); // Receive sensor data from Tiva on BBG UART4

                  if(recv > 0)
                  {

                        /* Send object detected message to Tiva if object
is detected */
                        if((strcmp(sensor.task_name,"DIST") == 0) &&
comm_rec.distance< 30)
                        {
                              pthread_mutex_lock(&lock_res);
                              uart_send(uart2, &obj_detect, sizeof(char));
                              pthread_mutex_unlock(&lock_res);

                              memset(buffer,'\0',sizeof(buffer));
```

```
                              sprintf(buffer,"WARN CN [%s] Object detected
sent from CN\n", time_stamp());
                              mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                        }

                        /* Send lux auto mode on to Tiva if lux is below
threshold */
                        else if((strcmp(sensor.task_name,"LUX") == 0) &&
comm_rec.lux < 10)
                        {

                              pthread_mutex_lock(&lock_res);
                              uart_send(uart2, &lux_auto, sizeof(char));
                              pthread_mutex_unlock(&lock_res);

                              memset(buffer,'\0',sizeof(buffer));
                              sprintf(buffer,"DEBUG CN [%s] Lux auto mode
sent from CN\n", time_stamp());
                              mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                        }

                        /* If water level sensor is inactive, or falls out
of range */
                        else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel > 3000)
                        {
                              water_outOfRange = 1;

                              memset(buffer,'\0',sizeof(buffer));
                              sprintf(buffer,"ERROR CN [%s] Water level
out of range\n", time_stamp());
                              mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);
                        }

                        /* Send valve closed to Tiva if water level is
below threshold */
                        else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel < 300 && already_closed == 0)
                        {
                              pthread_mutex_lock(&lock_res);
                              uart_send(uart2, &valve_close,
sizeof(char));
                              pthread_mutex_unlock(&lock_res);

                              memset(buffer,'\0',sizeof(buffer));
                              sprintf(buffer,"DEBUG CN [%s]  Valve close
sent from CN\n", time_stamp());
                              mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);

                              already_closed = 1;
```

```
                              already_open = 0;
                              water_outOfRange = 0;
                        }

                        /* Send valve open to Tiva if water level is above
threshold */
                        else if((strcmp(sensor.task_name,"WAT") == 0) &&
comm_rec.waterLevel >= 300 && already_open == 0)
                        {
                              pthread_mutex_lock(&lock_res);
                              uart_send(uart2, &valve_open, sizeof(char));
                              pthread_mutex_unlock(&lock_res);

                              memset(buffer,'\0',sizeof(buffer));
                              sprintf(buffer,"DEBUG CN [%s]  Valve open
sent from CN\n", time_stamp());
                              mq_send(msg_queue, buffer, MAX_BUFFER_SIZE,
0);

                              already_open = 1;
                              already_closed = 0;
                              water_outOfRange = 0;
                        }

                  }

            }
            uart_close(uart4); // Close UART4 file descriptor
            pthread_cancel(communication_thread);
      }
      return 0;
}
/**
 * @\file   log_receiver.c
 * @\author Sanju Prakash Kannioth and Steve Antony X
 * @\brief  This files contains the function definitions for tiva log
receiver module
 * @\date   03/30/2019
 *
 */

#include "log_receiver.h"


/**
--------------------------------------------------------------------------
-------------------
revecive_thread_callback
--------------------------------------------------------------------------
-------------------
*   This function is the thread callback function for logger messages
coming from Tiva
*
*   @\param         void
```

```
*
*   @\return        void
*
*/
void *revecive_thread_callback()
{
      char buffer[MAX_BUFFER_SIZE];


      memset(buffer,'\0',sizeof(buffer));
    sprintf(buffer,"DEBUG CN [%s] Tiva receive logger thread active\n",
time_stamp());
    mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

      printf("Inside receiver thread\n");

      /* Configure UART1 on BBG */
      uart_properties *uart1  = malloc(sizeof(uart_properties));
      uart1->uart_no = 1;
      uart1->baudrate = B115200;

      uint8_t isOpen1 = uart_config(uart1);

      char log[50];

      if(isOpen1 == 0)
      {
            memset(buffer,'\0',sizeof(buffer));
          sprintf(buffer,"DEBUG CN [%s] UART1 Opened successfully\n",
time_stamp());
          mq_send(msg_queue, buffer, MAX_BUFFER_SIZE, 0);

            while(1)
            {
                  memset(log,'\0', sizeof(log));

                  uart_receive_task(uart1,&log,sizeof(log));

            }
      }
      uart_close(uart1); // Close UART1
      return 0;
}
/*Reference : https://github.com/sijpesteijn/BBCLib */

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <termios.h>
#include <stdlib.h>
#include <stdint.h>
```

```
typedef enum {
      uart0 = 0, uart1 = 1, uart2 = 2, uart3 = 3, uart4 = 4, uart5 = 5
} uart;


typedef struct {
      int fd;
      uart uart_no;
      int baudrate;
}uart_properties;


uint8_t uart_config(uart_properties *uart)
{
      char path[15] = "/dev/ttyO";
      char uart_no[2];
      sprintf(uart_no,"%d",uart->uart_no);
      strcat(path,uart_no);

      struct termios uart_port;

      uart->fd = open(path, O_RDWR | O_NOCTTY | O_SYNC | O_NONBLOCK);
      if(uart->fd < 0) printf("port failed to open\n");

      uart_port.c_cflag = uart->baudrate | CS8 | CLOCAL | CREAD;
      uart_port.c_iflag&= ~(IGNBRK | ICRNL | INLCR | PARMRK | ISTRIP |
IXON); //= IGNPAR | ICRNL;
      uart_port.c_oflag = 0;
      uart_port.c_lflag = 0;

      uart_port.c_cc[VTIME] = 0;
      uart_port.c_cc[VMIN]  = 1;

      cfsetispeed(&uart_port, B115200);
      cfsetospeed(&uart_port, B115200);

      tcsetattr(uart->fd,TCSAFLUSH,&uart_port);
      printf("Opened\n");
      return 0;
}

int8_t uart_send(uart_properties *uart, char *tx, int length) {
      if (write(uart->fd, tx, length) == -1) {
            printf("Error in write\n");
            return -1;
      }
      printf("Wrote %s to uart %i\n", tx, uart->uart_no);
      return 0;
}


int8_t uart_receive(uart_properties *uart, char *rx, int length) {
      if (read(uart->fd, rx, length) == -1) {
            printf("Error in write\n");
```

```
            return -1;
      }

      printf("Read %s from uart %i\n", rx, uart->uart_no);
      return 0;
}

int8_t uart_close(uart_properties *uart) {
      close(uart->fd);
      return 0;
}

int main()
{

      uart_properties *uart  = malloc(sizeof(uart_properties));
      uart->uart_no = uart1;
      uart->baudrate = B115200;

      uart_properties *uart4 = malloc(sizeof(uart_properties));
      uart4->uart_no = 4;
      uart4->baudrate = B115200;

      uint8_t isOpen = uart_config(uart);
      uint8_t isOpen4 = uart_config(uart4);
      int i = 0;
      printf("Open success? %d\n", isOpen);
      if (isOpen == 0) {
            unsigned char receive[30];
            while(1)
            {
            //    printf("Entered while\n");
                  char buf[30];
                  sprintf(buf, "foo %d", ++i);

                  // Send data to uart1
                  if (uart_send(uart4, buf, strlen(buf) + 1) < 0) {
                        printf("Could not send data to uart port");
                        return -1;
                  }


                  usleep(100000);

                  uart_receive(uart4,receive,30);
                         sprintf(buf,"test uart 1");
                  uart_send(uart, buf, strlen(buf) + 1);
                  usleep(100000);
                  // Read data from uart1
                  if (uart_receive(uart, receive, 30)<  0) {
                        printf("Could not read data from uart port");
                        return -1;
                  }
            }
```

```
            uart_close(uart);
            uart_close(uart4);
      }
      printf("EOF\n");
      return 0;
}

/**
 * @\file   uart.c
 * @\author Sanju Prakash Kannioth
 * @\brief  This files contains the function definitions for uart
transmit and receive on BBG
 * @\date   04/29/2019
 * References : https://github.com/sijpesteijn/BBCLib,
 *
      https://en.wikibooks.org/wiki/Serial_Programming/termios
 *
 */

#include "uart.h"


char temp[MAX_BUFFER_SIZE];

struct sensor_struct *rx;

/**
------------------------------------------------------------------------
-------------------
uart_config
------------------------------------------------------------------------
-------------------
*   This function will configure the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                           success
*                           -1                              failure
*
*/

int8_t uart_config(uart_properties *uart)
{
      char path[15] = "/dev/ttyO";
      char uart_no[2];
      sprintf(uart_no,"%d",uart->uart_no);
      strcat(path,uart_no);

      struct termios uart_port;

      uart->fd = open(path, O_RDWR | O_NOCTTY | O_NDELAY);
      if(uart->fd < 0) printf("port failed to open\n");

      if(!isatty(uart->fd)) perror("Not a tty port\n");
```

```
	if(tcgetattr(uart->fd, &uart_port) < 0)
	{
		perror("Error getting attributes\n");
		return -1;

	}

	uart_port.c_cflag &= ~(CSIZE | PARENB);
	uart_port.c_cflag |= CLOCAL | CS8;

	uart_port.c_iflag&= ~(IGNBRK | ICRNL | INLCR | PARMRK | ISTRIP |
IXON);
	uart_port.c_oflag = 0;
	uart_port.c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN | ISIG);

	fcntl(uart->fd, F_SETFL, 0);

	if(cfsetispeed(&uart_port, B115200) < 0) // Set input baud rate
	{
		perror("Input baud rate invalid\n");
		return -1;
	}

	if(cfsetospeed(&uart_port, B115200) < 0) // Set output baud rate
	{
		perror("Output baud rate invalid\n");
		return -1;
	}

	if(tcsetattr(uart->fd,TCSANOW,&uart_port) < 0) // Change
configurations immediately
	{
		perror("Attribute setting invalid\n");
		return -1;
	}
	printf("Opened\n");
	return 0;
}

/**
------------------------------------------------------------------------
-------------------
uart_send
------------------------------------------------------------------------
-------------------
*   This function will send specified length of bytes over the UART on
BBG
*
*   @\param         uart_properties     specifies UART number
*                           tx                          byte to be sent
*                           length                      number of bytes
to be sent
```

```
*
*   @\return        -1                          Failure
*                           count                       Number of bytes
sent
*
*/
int8_t uart_send(uart_properties *uart, void *tx, int length) {
      int count = write(uart->fd, tx, length);
      if (count == -1) {
            printf("Error in write\n");
            return -1;
      }
      return count;
}



/**
-------------------------------------------------------------------------
-------------------
uart_receive
-------------------------------------------------------------------------
-------------------
*   This function will receive tiva sensor data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                           rx_r                    byte received
*                           length                      number of bytes
to be received
*
*   @\return        -1                          Failure
*                            count                      Number of bytes
sent
*
*/
int8_t uart_receive(uart_properties *uart, void *rx_r, int length) {
      int count = 0;
      count = read(uart->fd, rx_r, length);
      if (count  == -1) {
            return -1;
      }

      rx = rx_r;

      if(strcmp(rx->task_name,"DIST") == 0)
      {
            distance_active++;

            comm_rec.distance = rx->distance;

            comm_rec.mode = rx->mode;

      }
```

```
	else if(strcmp(rx->task_name,"LUX") == 0)
	{
		lux_active++;

		comm_rec.lux = rx->lux;

		comm_rec.mode = rx->mode;

	}

	else if((strcmp(rx->task_name,"WAT") == 0) && rx->water < 3000)
	{
		water_active++;

		water_outOfRange = 0;

		comm_rec.waterLevel = rx->water;

		comm_rec.mode = rx->mode;
	}

	else if(strcmp(rx->task_name,"BEA") == 0)
	{
		tiva_active++;
		comm_rec.dg_mode = rx->dg_mode;
	}

	return count;
}


/**
--------------------------------------------------------------------------
-------------------
uart_receive_task
--------------------------------------------------------------------------
-------------------
*   This function will receive tiva log data over the UART on BBG
*
*   @\param         uart_properties     specifies UART number
*                           rx_r                    byte received
*                           length                        number of bytes
to be received
*
*   @\return        -1                          Failure
*                            count                        Number of bytes
sent
*
*/
int8_t uart_receive_task(uart_properties *uart, void *rx_r, int length) {

	int count = 0;

	count = read(uart->fd, rx_r, length);
```

```
      if(count == -1)
            return -1;

      memset(temp,'\0', sizeof(temp));
      strcpy(temp,rx_r);
      printf("%s",temp);
      if(count > 1)
      {
            mq_send(msg_queue, temp, MAX_BUFFER_SIZE, 0);
      }

      return count;
}


/**
--------------------------------------------------------------------------------
-------------------
uart_close
--------------------------------------------------------------------------------
-------------------
*   This function will close the specific UART on BBG
*
*   @\param         uart_properties     specifies UART number
*
*   @\return        0                         success
*
*/
int8_t uart_close(uart_properties *uart) {
      close(uart->fd);
      return 0;
}
```