```
/*
 * delay.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef DELAY_H_
#define DELAY_H_

#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"

extern uint32_t g_sysClock;

#ifdef DEBUG
#ifndef DEBUG_ERROR
#define DEBUG_ERROR(x) if( ( x ) == pdTRUE ) { taskDISABLE_INTERRUPTS();
while(1); }
#endif
#else
#define DEBUG_ERROR(x)
#endif

static inline void DelayMs(uint32_t ms)
{
    MAP_SysCtlDelay((g_sysClock/ (1000 * 3))*ms);
}

static inline void DelayUs(uint32_t us)
{
    MAP_SysCtlDelay((g_sysClock/ (1000000 * 3))*us);
}


#endif /* DELAY_H_ */
/*
 * heartbeat.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef HEARTBEAT_H_
#define HEARTBEAT_H_


void heartbeat_start(uint32_t log_heartbeat_time_ms, uint32_t
led_heartbeat_time_ms);

#endif /* HEARTBEAT_H_ */
/*
 * dispatcher_task.h
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */
```

```
#ifndef INCLUDE_DISPATCHER_TASK_H_
#define INCLUDE_DISPATCHER_TASK_H_

#include <stdbool.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "queue.h"
#include "task.h"

//TODO: include a mutex lock in here to make the enque and notification
atomic. Let's see if needed
#define ENQUEUE_NOTIFY_DISPATCHER_TASK(comm_msg)    \
        ({ \
            uint8_t status = xQueueSend(getDispatcherQueueHandle(),
&comm_msg ,xMaxBlockTime);  \
            if(status == pdPASS) \
            {   \
                xTaskNotifyGive(getDispatcherTaskHandle());   \
            }   \
            status; \
        })

#define getDispatcherQueueHandle()           ({QueueHandle_t h =
DispatcherQueueHandle(NULL,1); h;})
#define setDispatcherQueueHandle(handle)
DispatcherQueueHandle(handle,0)

#define getDispatcherTaskHandle()           ({TaskHandle_t h =
DispatcherTaskHandle(NULL,1); h;})
#define setDispatcherTaskHandle(handle)
DispatcherTaskHandle(handle,0)

QueueHandle_t DispatcherQueueHandle(QueueHandle_t handle, bool get);
TaskHandle_t DispatcherTaskHandle(TaskHandle_t handle, bool get);


uint8_t DispatcherTask_init();


#endif /* INCLUDE_DISPATCHER_TASK_H_ */
/**
* @file - nordic_driver.h
* @brief - Header file for the driver functions of the NRF240L
*
* @author Gunj University of Colorado Boulder
* @date - 19th April 2017
**/

#ifndef __NORDIC_DRIVER_H__
#define __NORDIC_DRIVER_H__


#include <stdbool.h>
#include <stdint.h>

#include "inc/hw_memmap.h"
#include "driverlib/rom_map.h"
#include "driverlib/pin_map.h"
```

```
#include "driverlib/sysctl.h"

#include "driverlib/gpio.h"

#include "delay.h"

#define NORDIC_CE_SYSCTL_PORT       SYSCTL_PERIPH_GPIOE
#define NORDIC_CSN_SYSCTL_PORT      SYSCTL_PERIPH_GPIOE
#define NORDIC_IRQ_SYSCTL_PORT      SYSCTL_PERIPH_GPIOE

//#define NORDIC_CE_PORT  GPIO_PORTC_BASE
//#define NORDIC_CE_PIN   GPIO_PIN_4
//
//#define NORDIC_CSN_PORT GPIO_PORTC_BASE
//#define NORDIC_CSN_PIN  GPIO_PIN_5
//
//#define NORDIC_IRQ_PORT GPIO_PORTC_BASE
//#define NORDIC_IRQ_PIN  GPIO_PIN_6

#define NORDIC_CE_PORT  GPIO_PORTE_BASE
#define NORDIC_CE_PIN   GPIO_PIN_0

#define NORDIC_CSN_PORT GPIO_PORTE_BASE
#define NORDIC_CSN_PIN  GPIO_PIN_1

#define NORDIC_IRQ_PORT GPIO_PORTE_BASE
#define NORDIC_IRQ_PIN  GPIO_PIN_2

#define NORDIC_STATUS_RX_DR_MASK             (1<<6)
#define NORDIC_STATUS_TX_DS_MASK             (1<<5)
#define NORDIC_STATUS_MAX_RT_MASK           (1<<4)

typedef void (*NRF_INT_HANDLER_T)(void);

typedef enum{

      NRF_Mode_TX = 0,
      NRF_Mode_RX = 1

}NRF_Mode_t;

typedef enum{

    NRF_DR_1Mbps = 0,
    NRF_DR_2Mbps = 1

}NRF_DataRate_t;

typedef enum{

    NRF_PW_LOW = 0,
    NRF_PW_MED = 2,
    NRF_PW_HIGH = 3

}NRF_Power_t;

extern uint32_t g_sysClock;
```

```
/**
* @brief - Enable the chip select connection to Nordic
* @return void
**/
static inline void NRF_chip_enable()
{
      GPIOPinWrite(NORDIC_CSN_PORT, NORDIC_CSN_PIN, 0);
      DelayUs(50);
}

/**
* @brief - Disable the chip select connection to Nordic
* @return void
**/
static inline void NRF_chip_disable()
{
      GPIOPinWrite(NORDIC_CSN_PORT, NORDIC_CSN_PIN, NORDIC_CSN_PIN);
}

/**
* @brief - Enable TX/RX from the Nordic module
* @return void
**/
static inline void NRF_radio_enable()
{
    GPIOPinWrite(NORDIC_CE_PORT,NORDIC_CE_PIN, NORDIC_CE_PIN);
}

/**
* @brief - Disable TX/RX from the Nordic module
* @return void
**/
static inline void NRF_radio_disable()
{
    GPIOPinWrite(NORDIC_CE_PORT,NORDIC_CE_PIN, 0);
}

/**
* @brief - Initialize the NRF module
* Initialized the GPIO connections pertaining to the Nordic module
* @return void
**/
int8_t NRF_moduleInit(uint8_t use_interrupt, NRF_INT_HANDLER_T handler);

/**
* @brief - Disable the GPIO connections set up earlier for the Nordic
module
* @return void
**/
void NRF_moduleDisable();

/**
* @brief - Read a register from the NRF module
* @param - regAdd uint8_t
* @return uint8_t
**/
uint8_t NRF_read_register(uint8_t regAdd);
```

```
/**
* @brief - Write to a register from the NRF module
* @param - regAdd uint8_t
* @param - value uint8_t
* @return void
**/
void NRF_write_register(uint8_t regAdd, uint8_t value);

/**
* @brief - Write to the NRF module's status register
* @param - statusValue uint8_t
* @return void
**/
void NRF_write_status(uint8_t statusValue);

/**
* @brief - Read the NRF module's status register
* @return uint8_t
**/
uint8_t NRF_read_status();

/**
* @brief - Write to the NRF module's config register
* @param - configValue uint8_t
* @return void
**/
void NRF_write_config(uint8_t configValue);

/**
* @brief - Read the NRF module's config register
* @return uint8_t
**/
uint8_t NRF_read_config();


/**
* @brief - Read the NRF module's RF setup register
* @return uint8_t
**/
uint8_t NRF_read_rf_setup();

/**
* @brief - Write to the NRF module's RF setup register
* @param - rfStatusValue uint8_t
* @return void
**/
void NRF_write_rf_setup(uint8_t rfSetupValue);

/**
* @brief - Read the NRF module's RF CH register
* @return uint8_t
**/
uint8_t NRF_read_rf_ch();

/**
* @brief - Write to the NRF module's RF CH register
* @param - channel uint8_t
```

```
* @return void
**/
void NRF_write_rf_ch(uint8_t channel);

/**
* @brief - Reads 5 bytes of the NRF module's TX ADDR register
* @param - address uint8_t *
* @return void
**/
void NRF_read_TX_ADDR(uint8_t * address);

/**
* @brief - Writes 5 bytes of the NRF module's TX ADDR register
* @param - tx_addr uint8_t *
* @return void
**/
void NRF_write_TX_ADDR(uint8_t * tx_addr);

/**
* @brief - Read the NRF module's FIFO status register
* @return address uint8_t
**/
uint8_t NRF_read_fifo_status();

/**
* @brief - Send the command FLUSH_TX to the NRF module
* @return void
**/
void NRF_flush_tx_fifo();

/**
* @brief - Send the command FLUSH_RX to the NRF module
* @return void
**/
void NRF_flush_rx_fifo();

/**
* @brief - Send the activation command to the NRF module
* Activates the features: R_RX_PL_WID, W_ACK_PAYLOAD, W_TX_PAYLOAD_NOACK
* @return void
**/

void NRF_moduleSetup(NRF_DataRate_t DR, NRF_Power_t power);

void NRF_write_status(uint8_t statusValue);

uint8_t NRF_read_status();

void NRF_activate_cmd();

void NRF_read_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *address);
void NRF_write_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *rx_addr);

void NRF_write_En_AA(uint8_t data);
uint8_t NRF_read_En_AA();
void NRF_write_setup_retry(uint8_t data);
uint8_t NRF_read_setup_retry();
```

```
int8_t NRF_read_data(uint8_t *data, uint8_t len);
int8_t NRF_transmit_data(uint8_t *data, uint8_t len, uint8_t toRXMode);

void NRF_write_TXPayload(uint8_t *data, uint8_t len);
void NRF_TX_pulse();


void NRF_openReadPipe(uint8_t rx_pipe_number, uint8_t rx_addr[5], uint8_t
payload_size);

void NRF_openWritePipe(uint8_t tx_addr[5]);

void NRF_closeWritePipe();

void NRF_closeReadPipe(uint8_t rx_pipe_number);



#endif /* __NORDIC_DRIVER_H__ */
/*
 * communication_object.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef COMMUNICATION_OBJECT_H_
#define COMMUNICATION_OBJECT_H_

#include <string.h>

#ifndef BOARD_UID_SHIFT
#define BOARD_UID_SHIFT 24
#endif

#define GET_BOARD_UID_FROM_LOG_ID(id)    ((uint32_t)((id &
(0xFFU<<BOARD_UID_SHIFT))>>BOARD_UID_SHIFT))
#define GET_LOG_ID_FROM_LOG_ID(id)       ((id &
(~(0xFFU<<BOARD_UID_SHIFT))))


typedef enum
{
    MSG_ID_HEARTBEAT = 0,
    MSG_ID_MSG,
    MSG_ID_SENSOR_STATUS,
    MSG_ID_ERROR,
    MSG_ID_SENSOR_INFO,
    MSG_ID_INFO,
    MSG_ID_PICTURE,
    MSG_ID_OBJECT_DETECTED,
    MSG_ID_CLIENT_INFO_BOARD_TYPE,
    MSG_ID_CLIENT_INFO_UID,
    MSG_ID_CLIENT_INFO_CODE_VERSION,

    //The request id from the beaglebone
    MSG_ID_GET_SENSOR_STATUS,
    MSG_ID_GET_SENSOR_INFO,
```

```
    MSG_ID_GET_CLIENT_INFO_BOARD_TYPE,
    MSG_ID_GET_CLIENT_INFO_UID,
    MSG_ID_GET_CLIENT_INFO_CODE_VERSION,
    LAST_ID, //THIS ID IS JUST TO CALCULATE THE NUM OF IDS. THIS IS NOT
USED ANYWHERE This cannot be more than 255
}MSG_ID_T;

#define NUM_OF_ID   LAST_ID

const static char * const MSG_ID_STRING[NUM_OF_ID]  =
{
    "HEARTBEAT",
    "MSG",
    "STATUS",
    "ERROR",
    "INFO",
    "PICTURE",
    "OBJECT_DETECTED",
    "CLIENT_INFO_BOARD_TYPE",
    "CLIENT_INFO_UID",
    "CLIENT_INFO_CODE_VERSION",
    //The request id from the beaglebone
    "GET_SENSOR_STATUS",
    "GET_SENSOR_INFO",
    "GET_CLIENT_INFO_BOARD_TYPE",
    "GET_CLIENT_INFO_UID",
    "GET_CLIENT_INFO_CODE_VERSION",
};

//FOR DST and SRC Board ID
#define BBG_BOARD_ID        (0x00)
#define TIVA_BOARD1_ID      (0x01)
#define XYZ_TIVA_BOARD_ID   (0x02)

#define MY_TIVA_BOARD_ID    TIVA_BOARD1_ID

//For src and dst module ID
//Add all the modules' UID here for TIVA BOARD
#define TIVA_HEART_BEAT_MODULE   (1)
#define TIVA_SENSOR_MODULE       (2)
#define TIVA_CAMERA_MODULE       (3)
#define TIVA_COMM_MODULE         (4)

//Add all modules' UID here for BBG Board
#define BBG_LOGGER_MODULE       (1)
#define BBG_COMM_MODULE         (2)
#define BBG_SOCKET_MODULE       (3)
#define BBG_XYZ_MODULE          (4)

typedef uint8_t MSG_ID;
typedef uint8_t SRC_ID;
typedef uint8_t SRC_BOARD_ID;
typedef uint8_t DST_BOARD_ID;
typedef uint8_t DST_ID;

//This should be followed immediately by the PICTURE msg id
typedef struct cam_packet
{
```

```
    size_t length;
    void* frame;
}CAMERA_PACKET_T;

/*32byte LOG MESSAGE STRUCTURE*/
typedef struct COMM_MSG
{
    SRC_ID src_id;
    SRC_BOARD_ID src_brd_id;
    DST_ID dst_id;
    DST_BOARD_ID dst_brd_id;
    MSG_ID msg_id;
    union custom_data
    {
        float distance_cm;
        float sensor_value;
        CAMERA_PACKET_T *camera_packet;
        size_t nothing;
    }data;
    char message[18];
    uint16_t checksum;
}COMM_MSG_T;

static size_t COMM_MSG_SIZE = sizeof(COMM_MSG_T);

static uint16_t getCheckSum(const COMM_MSG_T *comm_msg)
{
    uint16_t checkSum = 0;
    uint8_t sizeOfPayload = sizeof(COMM_MSG_T) - sizeof(comm_msg-
>checksum);
    uint8_t *p_payload = (uint8_t*)comm_msg;
    int i;
    for(i = 0; i < sizeOfPayload; i++)
    {
        checkSum += *(p_payload+i);
    }
    return checkSum;
}

/*Return true if a match, return 0 is not a match*/
static inline uint8_t verifyCheckSum(const COMM_MSG_T *comm_msg)
{
    return (getCheckSum(comm_msg) == comm_msg->checksum);
}


#endif /* COMMUNICATION_OBJECT_H_ */
/*
 * communication_setup.h
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef INCLUDE_COMMUNICATION_SETUP_H_
#define INCLUDE_COMMUNICATION_SETUP_H_
```

```
void CommTask_init();


#endif /* INCLUDE_COMMUNICATION_SETUP_H_ */
/*
 * sonar_sensor.h
 *
 *  Created on: 28-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef INCLUDE_SONAR_SENSOR_H_
#define INCLUDE_SONAR_SENSOR_H_


void Sonar_sensor_init();
float sonarSensor_getDistance();

#endif /* INCLUDE_SONAR_SENSOR_H_ */
/*
 * comm_receiver_task.h
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef INCLUDE_COMM_RECEIVER_TASK_H_
#define INCLUDE_COMM_RECEIVER_TASK_H_


uint8_t CommReceiverTask_init();

#endif /* INCLUDE_COMM_RECEIVER_TASK_H_ */
/*
 * my_uart.h
 *
 *  Created on: 05-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef MY_UART_H_
#define MY_UART_H_

#ifndef __USE_FREERTOS
#define __USE_FREERTOS
#endif

#include <stdbool.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "semphr.h"

#include "driverlib/uart.h"

typedef enum UART_num
{
    UART_0 = 0,
    UART_1 = 1,
```

```
    UART_2 = 2,
    UART_3 = 3,
}UART_T;

/**
* @brief - Available Baud rates for the UART
*/
typedef enum BAUD_RATE
{
    BAUD_921600 = 921600,
    BAUD_460800 = 460800,
    BAUD_230400 = 230400,
    BAUD_115200 = 115200,
    BAUD_38400  = 38400,
    BAUD_57200  = 57200,
    BAUD_9600   = 9600,

}BAUD_RATE_ENUM;

extern const uint32_t UART[4];

#ifdef __USE_FREERTOS

xSemaphoreHandle g_pUARTMutex[4];

/* MACROS are threadsafe */
#define printf(fmt, ...)    xSemaphoreTake(g_pUARTMutex[UART_0],
portMAX_DELAY); UART0_printf(fmt, ##__VA_ARGS__);
xSemaphoreGive(g_pUARTMutex[UART_0])
#define puts(str)           xSemaphoreTake(g_pUARTMutex[UART_0],
portMAX_DELAY); UART0_putstr(str); xSemaphoreGive(g_pUARTMutex[UART_0])
#else

#define printf(fmt, ...)        UART0_printf(fmt, ##__VA_ARGS__)
#define puts(str)               UART0_putstr(str)
#define logger_log(ID, fmt, ...) UART0_printf(fmt, ##__VA_ARGS__)

#endif

#define UART_putchar(uart,ch)   (ch == '\n') ? UARTCharPut(UART[uart],
'\r'): 0; UARTCharPut(UART[uart], ch)
#define UART0_putchar(ch)     (ch == '\n') ? UARTCharPut(UART0_BASE,
'\r'): 0; UARTCharPut(UART0_BASE, ch)
#define UART3_putchar(ch)     (ch == '\n') ? UARTCharPut(UART3_BASE,
'\r'): 0; UARTCharPut(UART3_BASE, ch)

#define UART0_putstr(str)     UART_putstr(UART_0,str)
#define UART3_putstr(str)     UART_putstr(UART_3,str)

#define UART3_putRAW(p_data, len)    UART_putRAW(UART_3, p_data, len)

#define UART0_getRAW(p_data,len)    UART_getRAW(UART_0, p_data, len)
#define UART3_getRAW(p_data,len)    UART_getRAW(UART_3, p_data, len)

#define UART0_config(baudrate)  UART_config(UART_0, baudrate)
#define UART3_config(baudrate)  UART_config(UART_3, baudrate)

#define UART0_printf(fmt, ...)  UART_printf(UART_0,fmt, ##__VA_ARGS__)
```

```
#define UART3_printf(fmt, ...)  UART_printf(UART_3,fmt, ##__VA_ARGS__)

void UART_config(UART_T uart, BAUD_RATE_ENUM baudrate);
void UART_putstr(UART_T uart, const char *str);
void UART_printf(UART_T uart, const char *fmt, ...);
void UART_putRAW(UART_T uart, const uint8_t *data, size_t len);
size_t UART_getRAW(UART_T uart, uint8_t *data, size_t len);

#endif /* MY_UART_H_ */
/*
 * communication_interface.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef COMMUNICATION_INTERFACE_H_
#define COMMUNICATION_INTERFACE_H_

#include <stdbool.h>
#include <stdint.h>

#include "my_uart.h"
#include "nordic_driver.h"
#include "communication_object.h"

#define NRF_USE_INTERRUPT   (1)
#define NRF_NOTUSE_INTERRUPT   (0)

//#define COMM_TYPE_NRF
//#define RUN_TIME_SWITCH
#ifdef RUN_TIME_SWITCH
volatile uint8_t comm_type_uart = 1;

#define COMM_INIT()                 comm_init_NRF();
comm_init_UART(BAUD_115200)
void COMM_SEND(COMM_MSG_T comm_object)
{
    if(comm_type_uart)
    {
        comm_sendUART(comm_object);
    }
    else
    {
        comm_sendNRF(comm_object);
    }
}
#else
#ifdef COMM_TYPE_NRF
#define COMM_INIT(fd)               comm_init_NRF()
#define COMM_DEINIT(fd)             comm_deinit_NRF()
#define COMM_SEND(p_comm_object)    comm_sendNRF(p_comm_object)
#define COMM_SENDRAW(packet,len)    comm_sendNRF_raw(packet, len)
#define COMM_RECV(p_comm_object)    comm_recvNRF(p_comm_object);
#else
#define COMM_INIT()                 comm_init_UART()
//Will be used only on BBG
#define COMM_DEINIT(fd)             comm_deinit_UART(fd)
```

```
#define COMM_SEND(p_comm_object)    comm_sendUART(p_comm_object)
#define COMM_SENDRAW(packet,len)    comm_sendUARTRAW(packet,len)
#define COMM_RECV(p_comm_object)    comm_recvUART(p_comm_object)
#endif
#endif
#define RX_PIPE 1

//0x54,0x4d,0x52,0x68,0x7C
static uint8_t TXAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
static uint8_t RXAddr[5] = {0xC2,0xC2,0xC2,0xC2,0xC2};

#ifdef TIVA_BOARD
static inline void comm_init_UART()
{
    UART3_config(BAUD_921600);
}

static inline void comm_deinit_UART(int fd){}


static inline void comm_sendUARTRAW(uint8_t* packet, size_t len)
{
    UART3_putRAW(packet,len);
}

static inline void comm_sendUART(COMM_MSG_T *p_comm_object)
{
    UART3_putRAW((uint8_t*)p_comm_object, sizeof(COMM_MSG_T));
    /* This is needed to mark the end of send as the receiving side needs
the line termination as the BeagleBone has opened the UART is canonical
mode*/
    //UART3_putchar('\n');
}

static inline size_t comm_recvUART(COMM_MSG_T *p_comm_object)
{
    size_t ret = UART3_getRAW((uint8_t*)p_comm_object,
sizeof(COMM_MSG_T));
    return ret;
}

#else
//For BBG

static inline UART_FD_T comm_init_UART()
{
    return UART_Open(COM_PORT4);
}

static inline void comm_deinit_UART(UART_FD_T fd)
{
    UART_Close(fd);
}

static inline int32_t comm_sendUART(COMM_MSG_T *p_comm_object)
{
    return UART_putRAW((void*)p_comm_object,sizeof(COMM_MSG_T));
}
```

```
static inline int32_t comm_sendUARTRAW(COMM_MSG_T * comm_object, size_t
len)
{
    return UART_putRAW((void*)comm_object,len);
}

static inline int32_t comm_recvUART(COMM_MSG_T *comm_object)
{
    int32_t available = UART_dataAvailable(100);
    if(available == 1)
    {
        return UART_read((void*)comm_object,sizeof(COMM_MSG_T));
    }
    else
        return available;
}

#endif

//For BBG end


int8_t comm_init_NRF();
void comm_deinit_NRF();

int32_t comm_sendNRF_raw(uint8_t *data, size_t len);

//TODO:
int32_t comm_recvNRF_raw(uint8_t *data, size_t len);
int32_t comm_recvNRF(COMM_MSG_T *p_comm_object);

static inline int32_t comm_sendNRF(COMM_MSG_T *p_comm_object)
{
    return NRF_transmit_data((uint8_t*)(p_comm_object),
sizeof(COMM_MSG_T), true);
}


#endif /* COMMUNICATION_INTERFACE_H_ */
/*
 * cam_header.h
 *
 *  Created on: 29-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef CAM_HEADER_H_
#define CAM_HEADER_H_


#define MAX_FIFO_SIZE       0x5FFFF         //384KByte

#define RWBIT                                   0x80  //READ AND WRITE
BIT IS BIT[7]

#define ARDUCHIP_TEST1          0x00  //TEST register
```

```
#define ARDUCHIP_MODE           0x02  //Mode register
#define MCU2LCD_MODE            0x00
#define CAM2LCD_MODE            0x01
#define LCD2MCU_MODE            0x02

#define ARDUCHIP_TIM            0x03  //Timming control

//#define FIFO_PWRDN_MASK           0x20    //0 = Normal operation, 1 =
FIFO power down
//#define LOW_POWER_MODE              0x40      //0 = Normal mode,
1 = Low power mode

#define ARDUCHIP_FIFO           0x04  //FIFO and I2C control
#define FIFO_CLEAR_MASK         0x01
#define FIFO_START_MASK         0x02
#define FIFO_RDPTR_RST_MASK     0x10
#define FIFO_WRPTR_RST_MASK     0x20

#define ARDUCHIP_GPIO             0x06  //GPIO Write Register
#if !(defined (ARDUCAM_SHIELD_V2) || defined (ARDUCAM_SHIELD_REVC))
#define GPIO_RESET_MASK         0x01  //0 = Sensor reset,
1 =  Sensor normal operation
#define GPIO_PWDN_MASK          0x02  //0 = Sensor normal operation,    1
= Sensor standby
#define GPIO_PWREN_MASK         0x04    //0 = Sensor LDO disable,
1 = sensor LDO enable
#endif

#define BURST_FIFO_READ         0x3C  //Burst FIFO read operation
#define SINGLE_FIFO_READ        0x3D  //Single FIFO read operation

#define ARDUCHIP_REV            0x40  //ArduCHIP revision
#define VER_LOW_MASK            0x3F
#define VER_HIGH_MASK           0xC0

#define ARDUCHIP_TRIG           0x41  //Trigger source
#define VSYNC_MASK              0x01
#define SHUTTER_MASK            0x02
#define CAP_DONE_MASK           0x08

#define FIFO_SIZE1              0x42  //Camera write FIFO size[7:0] for
burst to read
#define FIFO_SIZE2              0x43  //Camera write FIFO size[15:8]
#define FIFO_SIZE3              0x44  //Camera write FIFO size[18:16]

#endif /* CAM_HEADER_H_ */
#ifndef OV2640_REGS_H
#define OV2640_REGS_H
//#include "ArduCAM.h"
//#include <avr/pgmspace.h>

#define OV2640_CHIPID_HIGH   0x0A
#define OV2640_CHIPID_LOW    0x0B


const uint8_t  OV2640_JPEG_INIT[192][2]  =
{
  { 0xff, 0x00 },
```

```
{ 0x2c, 0xff },
{ 0x2e, 0xdf },
{ 0xff, 0x01 },
{ 0x3c, 0x32 },
{ 0x11, 0x00 },
{ 0x09, 0x02 },
{ 0x04, 0x28 },
{ 0x13, 0xe5 },
{ 0x14, 0x48 },
{ 0x2c, 0x0c },
{ 0x33, 0x78 },
{ 0x3a, 0x33 },
{ 0x3b, 0xfB },
{ 0x3e, 0x00 },
{ 0x43, 0x11 },
{ 0x16, 0x10 },
{ 0x39, 0x92 },
{ 0x35, 0xda },
{ 0x22, 0x1a },
{ 0x37, 0xc3 },
{ 0x23, 0x00 },
{ 0x34, 0xc0 },
{ 0x36, 0x1a },
{ 0x06, 0x88 },
{ 0x07, 0xc0 },
{ 0x0d, 0x87 },
{ 0x0e, 0x41 },
{ 0x4c, 0x00 },
{ 0x48, 0x00 },
{ 0x5B, 0x00 },
{ 0x42, 0x03 },
{ 0x4a, 0x81 },
{ 0x21, 0x99 },
{ 0x24, 0x40 },
{ 0x25, 0x38 },
{ 0x26, 0x82 },
{ 0x5c, 0x00 },
{ 0x63, 0x00 },
{ 0x61, 0x70 },
{ 0x62, 0x80 },
{ 0x7c, 0x05 },
{ 0x20, 0x80 },
{ 0x28, 0x30 },
{ 0x6c, 0x00 },
{ 0x6d, 0x80 },
{ 0x6e, 0x00 },
{ 0x70, 0x02 },
{ 0x71, 0x94 },
{ 0x73, 0xc1 },
{ 0x12, 0x40 },
{ 0x17, 0x11 },
{ 0x18, 0x43 },
{ 0x19, 0x00 },
{ 0x1a, 0x4b },
{ 0x32, 0x09 },
{ 0x37, 0xc0 },
{ 0x4f, 0x60 },
{ 0x50, 0xa8 },
```

```
{ 0x6d, 0x00 },
{ 0x3d, 0x38 },
{ 0x46, 0x3f },
{ 0x4f, 0x60 },
{ 0x0c, 0x3c },
{ 0xff, 0x00 },
{ 0xe5, 0x7f },
{ 0xf9, 0xc0 },
{ 0x41, 0x24 },
{ 0xe0, 0x14 },
{ 0x76, 0xff },
{ 0x33, 0xa0 },
{ 0x42, 0x20 },
{ 0x43, 0x18 },
{ 0x4c, 0x00 },
{ 0x87, 0xd5 },
{ 0x88, 0x3f },
{ 0xd7, 0x03 },
{ 0xd9, 0x10 },
{ 0xd3, 0x82 },
{ 0xc8, 0x08 },
{ 0xc9, 0x80 },
{ 0x7c, 0x00 },
{ 0x7d, 0x00 },
{ 0x7c, 0x03 },
{ 0x7d, 0x48 },
{ 0x7d, 0x48 },
{ 0x7c, 0x08 },
{ 0x7d, 0x20 },
{ 0x7d, 0x10 },
{ 0x7d, 0x0e },
{ 0x90, 0x00 },
{ 0x91, 0x0e },
{ 0x91, 0x1a },
{ 0x91, 0x31 },
{ 0x91, 0x5a },
{ 0x91, 0x69 },
{ 0x91, 0x75 },
{ 0x91, 0x7e },
{ 0x91, 0x88 },
{ 0x91, 0x8f },
{ 0x91, 0x96 },
{ 0x91, 0xa3 },
{ 0x91, 0xaf },
{ 0x91, 0xc4 },
{ 0x91, 0xd7 },
{ 0x91, 0xe8 },
{ 0x91, 0x20 },
{ 0x92, 0x00 },
{ 0x93, 0x06 },
{ 0x93, 0xe3 },
{ 0x93, 0x05 },
{ 0x93, 0x05 },
{ 0x93, 0x00 },
{ 0x93, 0x04 },
{ 0x93, 0x00 },
{ 0x93, 0x00 },
{ 0x93, 0x00 },
```

```
{ 0x93, 0x00 },
{ 0x93, 0x00 },
{ 0x93, 0x00 },
{ 0x93, 0x00 },
{ 0x96, 0x00 },
{ 0x97, 0x08 },
{ 0x97, 0x19 },
{ 0x97, 0x02 },
{ 0x97, 0x0c },
{ 0x97, 0x24 },
{ 0x97, 0x30 },
{ 0x97, 0x28 },
{ 0x97, 0x26 },
{ 0x97, 0x02 },
{ 0x97, 0x98 },
{ 0x97, 0x80 },
{ 0x97, 0x00 },
{ 0x97, 0x00 },
{ 0xc3, 0xed },
{ 0xa4, 0x00 },
{ 0xa8, 0x00 },
{ 0xc5, 0x11 },
{ 0xc6, 0x51 },
{ 0xbf, 0x80 },
{ 0xc7, 0x10 },
{ 0xb6, 0x66 },
{ 0xb8, 0xA5 },
{ 0xb7, 0x64 },
{ 0xb9, 0x7C },
{ 0xb3, 0xaf },
{ 0xb4, 0x97 },
{ 0xb5, 0xFF },
{ 0xb0, 0xC5 },
{ 0xb1, 0x94 },
{ 0xb2, 0x0f },
{ 0xc4, 0x5c },
{ 0xc0, 0x64 },
{ 0xc1, 0x4B },
{ 0x8c, 0x00 },
{ 0x86, 0x3D },
{ 0x50, 0x00 },
{ 0x51, 0xC8 },
{ 0x52, 0x96 },
{ 0x53, 0x00 },
{ 0x54, 0x00 },
{ 0x55, 0x00 },
{ 0x5a, 0xC8 },
{ 0x5b, 0x96 },
{ 0x5c, 0x00 },
{ 0xd3, 0x00 },//{ 0xd3, 0x7f },
{ 0xc3, 0xed },
{ 0x7f, 0x00 },
{ 0xda, 0x00 },
{ 0xe5, 0x1f },
{ 0xe1, 0x67 },
{ 0xe0, 0x00 },
{ 0xdd, 0x7f },
{ 0x05, 0x00 },
```

```
  { 0x12, 0x40 },
  { 0xd3, 0x04 },//{ 0xd3, 0x7f },
  { 0xc0, 0x16 },
  { 0xC1, 0x12 },
  { 0x8c, 0x00 },
  { 0x86, 0x3d },
  { 0x50, 0x00 },
  { 0x51, 0x2C },
  { 0x52, 0x24 },
  { 0x53, 0x00 },
  { 0x54, 0x00 },
  { 0x55, 0x00 },
  { 0x5A, 0x2c },
  { 0x5b, 0x24 },
  { 0x5c, 0x00 },
  { 0xff, 0xff },
};

const uint8_t OV2640_YUV422[10][2]  =
{
  { 0xFF, 0x00 },
  { 0x05, 0x00 },
  { 0xDA, 0x10 },
  { 0xD7, 0x03 },
  { 0xDF, 0x00 },
  { 0x33, 0x80 },
  { 0x3C, 0x40 },
  { 0xe1, 0x77 },
  { 0x00, 0x00 },
  { 0xff, 0xff },
};

const uint8_t OV2640_JPEG[9][2]  =
{
  { 0xe0, 0x14 },
  { 0xe1, 0x77 },
  { 0xe5, 0x1f },
  { 0xd7, 0x03 },
  { 0xda, 0x10 },
  { 0xe0, 0x00 },
  { 0xFF, 0x01 },
  { 0x04, 0x08 },
  { 0xff, 0xff },
};

/* JPG 160x120 */
const uint8_t OV2640_160x120_JPEG[40][2]  =
{
  { 0xff, 0x01 },
  { 0x12, 0x40 },
  { 0x17, 0x11 },
  { 0x18, 0x43 },
  { 0x19, 0x00 },
  { 0x1a, 0x4b },
  { 0x32, 0x09 },
  { 0x4f, 0xca },
  { 0x50, 0xa8 },
  { 0x5a, 0x23 },
```

```
  { 0x6d, 0x00 },
  { 0x39, 0x12 },
  { 0x35, 0xda },
  { 0x22, 0x1a },
  { 0x37, 0xc3 },
  { 0x23, 0x00 },
  { 0x34, 0xc0 },
  { 0x36, 0x1a },
  { 0x06, 0x88 },
  { 0x07, 0xc0 },
  { 0x0d, 0x87 },
  { 0x0e, 0x41 },
  { 0x4c, 0x00 },
  { 0xff, 0x00 },
  { 0xe0, 0x04 },
  { 0xc0, 0x64 },
  { 0xc1, 0x4b },
  { 0x86, 0x35 },
  { 0x50, 0x92 },
  { 0x51, 0xc8 },
  { 0x52, 0x96 },
  { 0x53, 0x00 },
  { 0x54, 0x00 },
  { 0x55, 0x00 },
  { 0x57, 0x00 },
  { 0x5a, 0x28 },
  { 0x5b, 0x1e },
  { 0x5c, 0x00 },
  { 0xe0, 0x00 },
  { 0xff, 0xff },
};

/* JPG, 0x176x144 */

const uint8_t OV2640_176x144_JPEG[40][2]  =
{
  { 0xff, 0x01 },
  { 0x12, 0x40 },
  { 0x17, 0x11 },
  { 0x18, 0x43 },
  { 0x19, 0x00 },
  { 0x1a, 0x4b },
  { 0x32, 0x09 },
  { 0x4f, 0xca },
  { 0x50, 0xa8 },
  { 0x5a, 0x23 },
  { 0x6d, 0x00 },
  { 0x39, 0x12 },
  { 0x35, 0xda },
  { 0x22, 0x1a },
  { 0x37, 0xc3 },
  { 0x23, 0x00 },
  { 0x34, 0xc0 },
  { 0x36, 0x1a },
  { 0x06, 0x88 },
  { 0x07, 0xc0 },
  { 0x0d, 0x87 },
  { 0x0e, 0x41 },
```

```
  { 0x4c, 0x00 },
  { 0xff, 0x00 },
  { 0xe0, 0x04 },
  { 0xc0, 0x64 },
  { 0xc1, 0x4b },
  { 0x86, 0x35 },
  { 0x50, 0x92 },
  { 0x51, 0xc8 },
  { 0x52, 0x96 },
  { 0x53, 0x00 },
  { 0x54, 0x00 },
  { 0x55, 0x00 },
  { 0x57, 0x00 },
  { 0x5a, 0x2c },
  { 0x5b, 0x24 },
  { 0x5c, 0x00 },
  { 0xe0, 0x00 },
  { 0xff, 0xff },
};

/* JPG 320x240 */

const uint8_t OV2640_320x240_JPEG[40][2]  =
{
  { 0xff, 0x01 },
  { 0x12, 0x40 },
  { 0x17, 0x11 },
  { 0x18, 0x43 },
  { 0x19, 0x00 },
  { 0x1a, 0x4b },
  { 0x32, 0x09 },
  { 0x4f, 0xca },
  { 0x50, 0xa8 },
  { 0x5a, 0x23 },
  { 0x6d, 0x00 },
  { 0x39, 0x12 },
  { 0x35, 0xda },
  { 0x22, 0x1a },
  { 0x37, 0xc3 },
  { 0x23, 0x00 },
  { 0x34, 0xc0 },
  { 0x36, 0x1a },
  { 0x06, 0x88 },
  { 0x07, 0xc0 },
  { 0x0d, 0x87 },
  { 0x0e, 0x41 },
  { 0x4c, 0x00 },
  { 0xff, 0x00 },
  { 0xe0, 0x04 },
  { 0xc0, 0x64 },
  { 0xc1, 0x4b },
  { 0x86, 0x35 },
  { 0x50, 0x89 },
  { 0x51, 0xc8 },
  { 0x52, 0x96 },
  { 0x53, 0x00 },
  { 0x54, 0x00 },
  { 0x55, 0x00 },
```

```
  { 0x57, 0x00 },
  { 0x5a, 0x50 },
  { 0x5b, 0x3c },
  { 0x5c, 0x00 },
  { 0xe0, 0x00 },
  { 0xff, 0xff },
};



const uint8_t OV2640_640x480_JPEG2[40][2] =
{
      {0xff,0x01},          //001
      {0x11,0x01},          //002
      {0x12,0x00},          //003
      {0x17,0x11},          //004
      {0x18,0x75},          //005
      {0x32,0x36},          //006
      {0x19,0x01},          //007
      {0x1a,0x97},          //008
      {0x03,0x0f},          //009
      {0x37,0x40},          //010
      {0x4f,0xbb},          //011
      {0x50,0x9c},          //012
      {0x5a,0x57},          //013
      {0x6d,0x80},          //014
      {0x3d,0x34},          //015
      {0x39,0x02},          //016
      {0x35,0x88},          //017
      {0x22,0x0a},          //018
      {0x37,0x40},          //019
      {0x34,0xa0},          //020
      {0x06,0x02},          //021
      {0x0d,0xb7},          //022
      {0x0e,0x01},          //023
      {0xff,0x00},          //024
      {0xe0,0x04},          //025
      {0xc0,0xc8},          //026
      {0xc1,0x96},          //027
      {0x86,0x3d},          //028
      {0x50,0x89},          //029
      {0x51,0x90},          //030
      {0x52,0x2c},          //031
      {0x53,0x00},          //032
      {0x54,0x00},          //033
      {0x55,0x88},          //034
      {0x57,0x00},          //035
      {0x5a,0xa0},          //036
      {0x5b,0x78},          //037
      {0x5c,0x00},          //038
      {0xd3,0x04},          //039
      {0xe0,0x00},          //040
};

const uint8_t OV2640_640x480_JPEG[41][2] =
{
    {0xff, 0x01},
```

```
    {0x11, 0x01},
    {0x12, 0x00}, // Bit[6:4]: Resolution selection//0x02
    {0x17, 0x11}, // HREFST[10:3]
    {0x18, 0x75}, // HREFEND[10:3]
    {0x32, 0x36}, // Bit[5:3]: HREFEND[2:0]; Bit[2:0]: HREFST[2:0]
    {0x19, 0x01}, // VSTRT[9:2]
    {0x1a, 0x97}, // VEND[9:2]
    {0x03, 0x0f}, // Bit[3:2]: VEND[1:0]; Bit[1:0]: VSTRT[1:0]
    {0x37, 0x40},
    {0x4f, 0xbb},
    {0x50, 0x9c},
    {0x5a, 0x57},
    {0x6d, 0x80},
    {0x3d, 0x34},
    {0x39, 0x02},
    {0x35, 0x88},
    {0x22, 0x0a},
    {0x37, 0x40},
    {0x34, 0xa0},
    {0x06, 0x02},
    {0x0d, 0xb7},
    {0x0e, 0x01},
    {0xff, 0x00},
    {0xe0, 0x04},
    {0xc0, 0xc8},
    {0xc1, 0x96},
    {0x86, 0x3d},
    {0x50, 0x89},
    {0x51, 0x90},
    {0x52, 0x2c},
    {0x53, 0x00},
    {0x54, 0x00},
    {0x55, 0x88},
    {0x57, 0x00},
    {0x5a, 0xa0},
    {0x5b, 0x78},
    {0x5c, 0x00},
    {0xd3, 0x04},
    {0xe0, 0x00},
    {0xff, 0xff},
};

#endif
/**
* @file - spi.h
* @brief - Header file for the library functions for SPI
*
* @author Gunj University of Colorado Boulder
* @date - 19th April 2018
**/

#ifndef _SPI_H_
#define _SPI_H_

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
```

```
#include "driverlib/sysctl.h"

#include "driverlib/gpio.h"
#include "driverlib/ssi.h"
#include "driverlib/debug.h"

#include "my_uart.h"

#define SPI_1MZ 1000000
#define SPI_2MZ 2000000

/**
* @brief - Enum to allow flexibility of selection between SPI0 and SPI1
**/
typedef enum{
      SPI_0,
      SPI_1,
      SPI_2,
      SPI_3
}SPI_t;

typedef uint32_t SPI_Type;
typedef uint32_t SPI_SYSCTL_Type;

extern const SPI_Type SPI[4];

extern const SPI_SYSCTL_Type SPI_SYSCTL[4];

/**
* @brief - Initialize the GPIO pins associated with SPI
* Configure SPI in 3 wire mode and use a GPIO pin for chip select
* @param - spi SPI_t
* @return void
**/
void SPI_GPIO_init(SPI_t spi);

/**
* @brief - Enable the clock gate control for SPI
* @param - spi SPI_t
* @return void
**/
static inline void SPI_clock_init(SPI_t spi, uint32_t g_sysclock)
{
    MAP_SysCtlPeripheralEnable(SPI_SYSCTL[spi]);
    uint32_t src = SSIClockSourceGet(SPI[spi]);
    if(src == SSI_CLOCK_SYSTEM)
    {
        printf("SSI Using System Clock\n");
    }
    else if(src == SSI_CLOCK_PIOSC)
    {
        printf("SSI Using PIOSC\n");
    }

    //SSIAdvModeSet(SPI[spi], SSI_ADV_MODE_LEGACY);

    SSIConfigSetExpClk(SPI[spi], g_sysclock, SSI_FRF_MOTO_MODE_0,
                           SSI_MODE_MASTER, SPI_1MZ, 8);
```

```
}



/**
* @brief - Perform the initialization routine for the SPI module
* @param - spi SPI_t
* @return void
**/
static inline void SPI_init(SPI_t spi /*, uint32_t g_sysclock*/)
{
//    SSIConfigSetExpClk(SPI[spi], g_sysclock, SSI_FRF_MOTO_MODE_0,
SSI_MODE_MASTER, SPI_1MZ, 8);
    SPI_GPIO_init(spi);
    SSIEnable(SPI[spi]);
}

/**
* @brief - Disable the GPIO pins earlier initialized for the SPI module
* @return void
**/
static inline void SPI_disable(SPI_t spi)
{
    SSIDisable(SPI[spi]);
    MAP_SysCtlPeripheralDisable(SPI_SYSCTL[spi]);
}

/**
* @brief - Blocks until SPI transmit buffer has completed transmitting
* @param - spi SPI_t
* @return void
**/
static inline void SPI_flush(SPI_t spi)
{
    while(SSIBusy(SPI[spi]));
}

static inline void SPI_flushRXFIFO(SPI_t spi)
{
    uint32_t garbage;
    while(SSIDataGetNonBlocking(SPI[spi], &garbage));
}


/**
* @brief - Read a single byte from the SPI bus
* @param - spi SPI_t
* @return uint8_t
**/
static inline uint8_t SPI_read_byte(SPI_t spi)
{
    uint32_t data;
    SSIDataGet(SPI[spi], &data);
    return ((uint8_t)(data & 0xFF));
}

/**
```

```
* @brief - Read a single byte from the SPI bus without waiting
* @param - spi SPI_t
* @return uint8_t
**/
static inline uint8_t SPI_read_byte_NonBlocking(SPI_t spi)
{

    uint32_t data;
    SSIDataGetNonBlocking(SPI[spi], &data);
    return ((uint8_t)(data & 0xFF));
}

/**
* @brief - Write a single byte on to the SPI bus
* @param - spi SPI_t
* @param - byte uint8_t
* @return void
**/
static inline void SPI_write_byte(SPI_t spi, uint8_t byte)
{
    SSIDataPut(SPI[spi],((uint32_t)byte & 0x000000FF));
    SPI_flush(spi);
}

/**
* @brief - Write a single byte on to the SPI bus without blocking
* @param - spi SPI_t
* @param - byte uint8_t
* @return void
**/
static inline void SPI_write_byte_NonBlocking(SPI_t spi, uint8_t byte)
{

    SPI_flush(spi);
    SSIDataPutNonBlocking(SPI[spi],((uint32_t)byte & 0x000000FF));
    SPI_flush(spi);
}

/**
* @brief - Send a packet on to the SPI bus
* Send multiple bytes given a pointer to an array and the number of bytes
to be sent
* @param - spi SPI_t
* @param - p uint8_t
* @param - length size_t
* @return void
**/
void SPI_write_packet(SPI_t spi, uint8_t* p, size_t length);

/**
* @brief - Read a packet from the SPI bus
* Read multiple bytes given a pointer to an array for storage and the
number of bytes to be read
* @param - spi SPI_t
* @param - p uint8_t *
* @param - length size_t
* @return void
**/
```

```
void SPI_read_packet(SPI_t spi, uint8_t* p, size_t length);


#endif /* SOURCES_SPI0_H_ */
/*
 * ultrasonic_sensor_task.h
 *
 *  Created on: 27-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef INCLUDE_SONAR_SENSOR_TASK_H_
#define INCLUDE_SONAR_SENSOR_TASK_H_

#include <stdbool.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "queue.h"
#include "semphr.h"
#include "task.h"
#include <string.h>

#define EVENT_SONAR_PERIODIC_UPDATEVAL   ((0x01)<<0)
#define EVENT_SONAR_REQUEST_GETVAL       ((0x01)<<1)
#define EVENT_SONAR_SENSOR_INFO          ((0x01)<<2)

//Handy macros
#define NOTIFY_SONAR_SENSOR_TASK(EVENT_ID)
xTaskNotify(getSonar_sensorTaskHandle(),EVENT_ID,eSetBits)

#define ENQUEUE_NOTIFY_SONAR_SENSOR_TASK(comm_msg, EVENT_ID)    \
        ({ \
            uint8_t status = xQueueSend(getSonar_sensorQueueHandle(),
&comm_msg ,xMaxBlockTime);  \
            if(status == pdPASS) \
            {   \

xTaskNotify(getSonar_sensorTaskHandle(),EVENT_ID,eSetBits); \
            }   \
            status; \
        })

#define getSonar_sensorQueueHandle()            ({QueueHandle_t h =
Sonar_sensorQueueHandle(NULL,1); h;})
#define setSonar_sensorQueueHandle(handle)
Sonar_sensorQueueHandle(handle,0)

#define getSonar_sensorTaskHandle()           ({TaskHandle_t h =
Sonar_sensorTaskHandle(NULL,1); h;})
#define setSonar_sensorTaskHandle(handle)
Sonar_sensorTaskHandle(handle,0)

QueueHandle_t Sonar_sensorQueueHandle(QueueHandle_t handle, bool get);
TaskHandle_t Sonar_sensorTaskHandle(TaskHandle_t handle, bool get);

uint8_t SonarSensorTask_init();

#endif /* INCLUDE_SONAR_SENSOR_TASK_H_ */
```

```
/*
 * comm_sender_task.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef COMM_SENDER_TASK_H_
#define COMM_SENDER_TASK_H_

#include <stdbool.h>
#include <stdint.h>
#include "FreeRTOS.h"
#include "queue.h"
#include "semphr.h"
#include "task.h"
#include <string.h>
#include "communication_object.h"
#include "my_uart.h"

#define EVENT_COMM_SENDER_HEARTBEAT      ((0x01)<<0)
#define EVENT_COMM_SENDER_MSG            ((0x01)<<1)
#define EVENT_COMM_SENDER_STATUS         ((0x01)<<2)
#define EVENT_COMM_SENDER_ERROR          ((0x01)<<3)
#define EVENT_COMM_SENDER_INFO           ((0x01)<<4)
#define EVENT_COMM_SENDER_BOARD_TYPE     ((0x01)<<5)
#define EVENT_COMM_SENDER_UID            ((0x01)<<6)
#define EVENT_COMM_SENDER_CODE_VERSION   ((0x01)<<7)
#define EVENT_COMM_SENDER_PICTURE        ((0x01)<<8)
#define EVENT_COMM_SENDER_OBJECT_DETECTED   ((0x01)<<9)
//Handy macros
#define NOTIFY_COMM_OBJECT(EVENT_ID)
xTaskNotify(getComm_senderTaskHandle(),EVENT_ID,eSetBits)

#define COMM_CREATE_OBJECT(name, src_board_id, source_id, dest_id)
COMM_MSG_T name = { .src_brd_id = src_board_id, .src_id = source_id,
.dst_id = dest_id, .dst_brd_id = BBG_BOARD_ID }
#define COMM_OBJECT_MSGID(comm_msg,msgid)   comm_msg.msg_id = msgid
#define FILL_CHECKSUM(p_comm_msg)           do{ (p_comm_msg)->checksum =
getCheckSum(p_comm_msg); }while(0)
#define COMM_FILL_MSG(comm_msg,p_str)
strncpy(comm_msg.message,p_str,sizeof(comm_msg.message))


SemaphoreHandle_t xCOMM_SENDER_NOTIFY_MUTEX;

//TODO: include a mutex lock in here to make the enque and notification
atomic. Let's see if needed
#define ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg, EVENT_ID)    \
        ({ \
            uint8_t status = xSemaphoreTake(xCOMM_SENDER_NOTIFY_MUTEX,
portMAX_DELAY); \
            if(status == pdTRUE) \
            {status = xQueueSend(getComm_senderQueueHandle(), &comm_msg
,pdMS_TO_TICKS(500)); \
            if(status == pdPASS) \
            {   \
```

```
xTaskNotify(getComm_senderTaskHandle(),EVENT_ID,eSetBits);   \
            }   \
            xSemaphoreGive(xCOMM_SENDER_NOTIFY_MUTEX); }\
            else    \
            { printf("SemTake Handle error. %s\n",__FUNCTION__);}   \
            status; \
        })

#define getComm_senderQueueHandle()           ({QueueHandle_t h =
Comm_senderQueueHandle(NULL,1); h;})
#define setComm_senderQueueHandle(handle)
Comm_senderQueueHandle(handle,0)

#define getComm_senderTaskHandle()           ({TaskHandle_t h =
Comm_senderTaskHandle(NULL,1); h;})
#define setComm_senderTaskHandle(handle)
Comm_senderTaskHandle(handle,0)

QueueHandle_t Comm_senderQueueHandle(QueueHandle_t handle, bool get);
TaskHandle_t Comm_senderTaskHandle(TaskHandle_t handle, bool get);

uint8_t CommSenderTask_init();


#endif /* COMM_SENDER_TASK_H_ */
/*
 * camera_interface.h
 *
 *  Created on: 29-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef INCLUDE_CAMERA_INTERFACE_H_
#define INCLUDE_CAMERA_INTERFACE_H_



void CameraInit();
uint32_t SendFrame();
void I2C_init();

/* Ported functions from arducam arduino library*/
void wrSensorReg8_8(uint8_t reg, uint8_t data);
uint8_t rdSensorReg8_8(uint8_t reg);
void wrSensorRegs8_8(uint8_t **array_reg_value_pair);
uint8_t get_bit(uint8_t addr, uint8_t bit);
uint32_t read_fifo_length();
uint8_t read_reg(uint8_t reg);
void write_reg(uint8_t reg, uint8_t value);
uint8_t read_fifo_burst();
uint8_t transfer(uint8_t val);


static inline void write(uint8_t data)
{
//    UARTCharPut(UART0_BASE, data);
    UARTCharPut(UART3_BASE, data);
```

```
}

static inline void CS_LOW()
{
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
}

static inline void CS_HIGH()
{
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
}



#endif /* INCLUDE_CAMERA_INTERFACE_H_ */
/*
 * application.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef APPLICATION_H_
#define APPLICATION_H_

void application_run();

#endif /* APPLICATION_H_ */
/*
 * communication_setup.c
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */

#include "my_uart.h"
#include "comm_sender_task.h"
#include "comm_receiver_task.h"
#include "dispatcher_task.h"

void CommTask_init()
{
    if(CommSenderTask_init())
    {
        printf("[ERROR] %s\n",__FUNCTION__);
        while(1);
    }

    if(CommReceiverTask_init())
    {
        printf("[ERROR] %s\n",__FUNCTION__);
        while(1);
    }

    if(DispatcherTask_init())
    {
        printf("[ERROR] %s\n",__FUNCTION__);
        while(1);
```

```
    }
}
/*
 * dispatcher_task.c
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */

#include "priorities.h"

#include "dispatcher_task.h"
#include "communication_object.h"
#include "comm_sender_task.h"
#include "sonar_sensor_task.h"
#include "my_uart.h"

#define DISPATCHER_QUEUE_ITEMSIZE   (sizeof(COMM_MSG_T))
#define DISPATCHER_QUEUE_LENGTH     20


volatile uint8_t dispatcherTaskInitDone = 0;
static QueueHandle_t h_dispatcherQueue;
static TaskHandle_t h_dispatcherTask;

QueueHandle_t DispatcherQueueHandle(QueueHandle_t handle, bool get)
{
    if(get)
        return h_dispatcherQueue;
    else
    {
        h_dispatcherQueue = handle;
        return h_dispatcherQueue;
    }
}

TaskHandle_t DispatcherTaskHandle(TaskHandle_t handle, bool get)
{
    if(get)
        return h_dispatcherTask;
    else
    {
        h_dispatcherTask = handle;
        return h_dispatcherTask;
    }
}

/* Create the entry function */
static void dispatcher_task_entry(void *params)
{
    /* Waits on the notification from comm_recv and deq comm item from
the queue, process it depending on the msg id and dst id */
    /* Call function accordingly */
    const TickType_t xMaxBlockTime = pdMS_TO_TICKS(5000);
    BaseType_t xResult;
    COMM_MSG_T comm_msg;
    while(1)
    {
```

```
        /* Wait to be notified of an interrupt. */
        xResult = ulTaskNotifyTake( pdFALSE,    /* Using as counting
semaphore. */
                                    portMAX_DELAY);

        if( xResult == pdPASS )
        {
            /* A Signal was received.  Dequeue the comm_msg from task
queue */
            if(h_dispatcherQueue &&
xQueueReceive(h_dispatcherQueue,&comm_msg, xMaxBlockTime))
            {
                if(!verifyCheckSum(&comm_msg))
                {
                    printf("Checksum error\n");
                    continue;
                }
                printf("DISPATCHING: %s\n",comm_msg.message);
                /* Process the comm msg. Decide on which parameter do we
need to dispatch it*/
                if(comm_msg.dst_id == TIVA_COMM_MODULE)
                {
                    switch(comm_msg.msg_id)
                    {
                    case MSG_ID_GET_CLIENT_INFO_BOARD_TYPE:
                        printf("GET CLIENT INFO BOARD TYPE\n");
                        NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_BOARD_TYPE);
                        break;
                    case MSG_ID_GET_CLIENT_INFO_CODE_VERSION:
                        printf("GET CLIENT INFO CODE VERSION\n");

NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_CODE_VERSION);
                        break;
                    case MSG_ID_GET_CLIENT_INFO_UID:
                        printf("GET CLIENT INFO UID\n");
                        NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_UID);
                        break;
                    default:
                        printf("Invalid Msg Id:%d from BOARD ID:
%d\n",comm_msg.msg_id,comm_msg.src_brd_id);
                        break;
                    }

                }
                else if(comm_msg.dst_id == TIVA_SENSOR_MODULE)
                {
                    if(comm_msg.msg_id == MSG_ID_GET_SENSOR_STATUS)
                    {
                        ENQUEUE_NOTIFY_SONAR_SENSOR_TASK(comm_msg,
EVENT_SONAR_REQUEST_GETVAL);
                    }
                    else if(comm_msg.msg_id == MSG_ID_GET_SENSOR_INFO)
                    {
                        ENQUEUE_NOTIFY_SONAR_SENSOR_TASK(comm_msg,
EVENT_SONAR_SENSOR_INFO);
                    }
                    else
                    {
```

```
COMM_CREATE_OBJECT(send_comm_msg,MY_TIVA_BOARD_ID,TIVA_SENSOR_MODULE,comm
_msg.src_id);
                        send_comm_msg.msg_id = MSG_ID_ERROR;
                        send_comm_msg.data.distance_cm = 0;
                        COMM_FILL_MSG(send_comm_msg,"Invalid Request");

ENQUE_NOTIFY_COMM_SENDER_TASK(send_comm_msg,EVENT_COMM_SENDER_STATUS);
                    }
                }
                else if(comm_msg.dst_id == TIVA_CAMERA_MODULE)
                {
                    if(comm_msg.msg_id == MSG_ID_GET_SENSOR_STATUS)
                    {

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_CAMERA_MODULE,BBG_LOGGE
R_MODULE);
                        comm_msg.msg_id = MSG_ID_SENSOR_STATUS;
                        comm_msg.data.sensor_value = 0.12;
                        COMM_FILL_MSG(comm_msg,"160x140/jpeg");

ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg,EVENT_COMM_SENDER_STATUS);
                    }
                    else if(comm_msg.msg_id == MSG_ID_GET_SENSOR_INFO)
                    {

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_CAMERA_MODULE,BBG_LOGGE
R_MODULE);
                        comm_msg.msg_id = MSG_ID_SENSOR_INFO;
                        COMM_FILL_MSG(comm_msg,"ArduCAM/jpeg");

ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg,EVENT_COMM_SENDER_STATUS);
                    }
                }
                else
                {
                    printf("INVALID MODULE ID\n");
                }
            }
            else
            {
                printf("[Error] Q RECV %s\n",__FUNCTION__);
            }
        }
//        else
//        {
//            printf("DISPATCHER NOTIFICATION: TIMEOUT\n");
//        }
    }
}

uint8_t DispatcherTask_init()
{
    /* Creating a Queue required for getting the comm msg recv from comm
recv task */
    QueueHandle_t h_dispatcherQ = xQueueCreate(DISPATCHER_QUEUE_LENGTH,
DISPATCHER_QUEUE_ITEMSIZE);
    setDispatcherQueueHandle(h_dispatcherQ);
```

```
    TaskHandle_t h_dispatcherTask;

    /* Create the task*/
    if(xTaskCreate(dispatcher_task_entry, (const portCHAR *)"Dispatcher
Task", 128, NULL,
                        tskIDLE_PRIORITY + PRIO_DISPATCHERTASK,
&h_dispatcherTask) != pdTRUE)
    {
        return (1);
    }

    //Setting the dispatcher task handle for future use
    setDispatcherTaskHandle(h_dispatcherTask);
    /* Return the createtask ret value */
    return 0;

}


/*
 * application_hooks.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef APPLICATION_HOOKS_C_
#define APPLICATION_HOOKS_C_

#include <stdint.h>
#include <stdbool.h>
#include "FreeRTOS.h"
#include "task.h"
#include "my_uart.h"

void vApplicationStackOverflowHook(xTaskHandle *pxTask, char *pcTaskName)
{
    //
    // This function can not return, so loop forever.  Interrupts are
disabled
    // on entry to this function, so no processor interrupts will
interrupt
    // this loop.
    //
    //TODO: notify logging task
    printf("\nSTACK ERROR - TASK: %s\n",pcTaskName);
    while(1)
    {
    }
}

void vApplicationMallocFailedHook( void )
{
    //TODO: notify logging task
    printf("\nMALLOC ERROR\n");
    while(1)
    {
```

```
    }
}

#endif /* APPLICATION_HOOKS_C_ */
/*
 * application.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

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

#include "inc/hw_ints.h"
#include "driverlib/rom_map.h"
#include "driverlib/rom.h"
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#include "FreeRTOS.h"
#include "task.h"
//#include "queue.h"
//#include "semphr.h"

#include "my_uart.h"
#include "heartbeat.h"
#include "application.h"
#include "communication_setup.h"
#include "comm_sender_task.h"
#include "sonar_sensor_task.h"
#include "camera_interface.h"

//#define CLOCK_FREQ 120000000
#define CLOCK_FREQ 16000000
uint32_t g_sysClock = CLOCK_FREQ;


void send_boardIdentification()
{
    NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_BOARD_TYPE);
    NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_UID);
    NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_CODE_VERSION);
}

void application_run()
{
    // Set the clocking to run directly from the crystal at 120MHz.
   g_sysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                                        SYSCTL_OSC_MAIN |
                                        SYSCTL_USE_PLL |
                                        SYSCTL_CFG_VCO_480),
g_sysClock);

//   UART0_config(BAUD_115200);
   UART0_config(BAUD_921600);
```

```
    ROM_IntMasterEnable();
    printf("\n----- GUNJ Project2 --------\n");

    CommTask_init();
    CameraInit();

    send_boardIdentification();

    heartbeat_start(1000, 500);

    if(SonarSensorTask_init())
    {
        printf("[ERROR] %s\n",__FUNCTION__);
        while(1);
    }


    printf("SUCCESS - All tasks are created. Starting scheduler....\n");

    vTaskStartScheduler();

    while(1);


}
/*
 * sonar_sensor_task.c
 *
 *  Created on: 27-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdint.h>
#include <stdbool.h>
#include "FreeRTOS.h"
#include "timers.h"
#include "priorities.h"
#include <limits.h>

#include "sonar_sensor_task.h"
#include "comm_sender_task.h"
#include "communication_object.h"
#include "sonar_sensor.h"
#include "delay.h"

#define DISTANCE_THRESHOLD_CM   10

#define SONAR_SENSOR_QUEUE_ITEMSIZE   (sizeof(COMM_MSG_T))
#define SONAR_SENSOR_QUEUE_LENGTH     20

volatile uint8_t TaskInitDone = 0;

static TaskHandle_t h_sonar_sensorTask;
static QueueHandle_t h_sonar_sensorQueue;

static float current_sensor_distance_cm = 0;
static uint8_t object_detected = 0;
```

```
QueueHandle_t Sonar_sensorQueueHandle(QueueHandle_t handle, bool get)
{
    if(get)
        return h_sonar_sensorQueue;
    else
    {
        h_sonar_sensorQueue = handle;
        return h_sonar_sensorQueue;
    }
}

TaskHandle_t Sonar_sensorTaskHandle(TaskHandle_t handle, bool get)
{
    if(get)
        return h_sonar_sensorTask;
    else
    {
        h_sonar_sensorTask = handle;
        return h_sonar_sensorTask;
    }
}

static void send_sonarSensorValue(uint8_t dst_board_id, uint8_t
dst_module_id)
{

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_SENSOR_MODULE,dst_modul
e_id);
    comm_msg.dst_brd_id = dst_board_id;
    comm_msg.msg_id = MSG_ID_SENSOR_STATUS;
    comm_msg.data.distance_cm = current_sensor_distance_cm;
    COMM_FILL_MSG(comm_msg,"Distance in cm");
    ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg,EVENT_COMM_SENDER_STATUS);
}

static void send_sonarSensorInfo(uint8_t dst_board_id, uint8_t
dst_module_id)
{

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_SENSOR_MODULE,dst_modul
e_id);
    comm_msg.dst_brd_id = dst_board_id;
    comm_msg.msg_id = MSG_ID_SENSOR_INFO;
    comm_msg.data.distance_cm = current_sensor_distance_cm;
    COMM_FILL_MSG(comm_msg,"Sonar/unit:cm/1s");
    ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg,EVENT_COMM_SENDER_INFO);
}

static void send_sonarObjectDetected()
{

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_SENSOR_MODULE,BBG_LOGGE
R_MODULE);
    comm_msg.msg_id = MSG_ID_OBJECT_DETECTED;
    comm_msg.data.distance_cm = current_sensor_distance_cm;
    COMM_FILL_MSG(comm_msg,"Sonar/Th:10cm");
```

```
ENQUE_NOTIFY_COMM_SENDER_TASK(comm_msg,EVENT_COMM_SENDER_OBJECT_DETECTED)
;
}


/* Create the entry task*/
static void sonar_sensor_task_entry(void *params)
{
        const TickType_t xMaxBlockTime = pdMS_TO_TICKS(500);
        BaseType_t xResult;
        uint32_t notifiedValue = 0;
        while(1)
        {
            /* Wait to be notified of an interrupt. */
            xResult = xTaskNotifyWait( pdFALSE,    /* Don't clear bits on
entry. */
                                ULONG_MAX,        /* Clear all bits on
exit. */
                                &notifiedValue, /* Stores the notified
value. */
                                portMAX_DELAY);

            if( xResult == pdPASS )
            {
                if(notifiedValue & EVENT_SONAR_PERIODIC_UPDATEVAL)
                {
                    //Perform Measurement
                    current_sensor_distance_cm =
sonarSensor_getDistance();
//                    printf("Distance:
%f\n",current_sensor_distance_cm);
                    (object_detected%5 == 0)  ? object_detected = 0 :
object_detected++;
                    if(!object_detected && (current_sensor_distance_cm >
0) && (current_sensor_distance_cm < DISTANCE_THRESHOLD_CM))
                    {
                        //Notify object detected
                        send_sonarObjectDetected();
                        object_detected = 1;
                    }
                }

                if(notifiedValue & EVENT_SONAR_REQUEST_GETVAL)
                {
                    COMM_MSG_T comm_msg = {0};
                    if(h_sonar_sensorQueue &&
xQueueReceive(h_sonar_sensorQueue,&comm_msg, xMaxBlockTime))
                    {
                        send_sonarSensorValue(comm_msg.src_brd_id,
comm_msg.src_id);
                    }
                    else
                    {
                        printf("SONAR QUEUE Timeout\n");
                    }

                }
```

```
                if(notifiedValue & EVENT_SONAR_SENSOR_INFO)
                {
                    COMM_MSG_T comm_msg = {0};
                    if(h_sonar_sensorQueue &&
xQueueReceive(h_sonar_sensorQueue,&comm_msg, xMaxBlockTime))
                    {
                        send_sonarSensorInfo(comm_msg.src_brd_id,
comm_msg.src_id);
                    }
                    else
                    {
                        printf("SONAR QUEUE Timeout\n");
                    }
                }
            }
//            else
//            {
//                printf("SENSOR NOTIFICATION: TIMEOUT\n");
//            }
        }


}

void vPeriodicUpdateTimerCallback(TimerHandle_t h_timer)
{
    NOTIFY_SONAR_SENSOR_TASK(EVENT_SONAR_PERIODIC_UPDATEVAL);
}


uint8_t SonarSensorTask_init()
{
    Sonar_sensor_init();
    TaskHandle_t h_Task;
    QueueHandle_t h_sonar_sensorQ =
xQueueCreate(SONAR_SENSOR_QUEUE_LENGTH, SONAR_SENSOR_QUEUE_ITEMSIZE);
    setSonar_sensorQueueHandle(h_sonar_sensorQ);


    TimerHandle_t periodic_getDistance_timer =
xTimerCreate("PERIODIC_GET_DISTANCE", pdMS_TO_TICKS(1000) , pdTRUE,
(void*)0, vPeriodicUpdateTimerCallback);
    DEBUG_ERROR(periodic_getDistance_timer == NULL);


    if(xTaskCreate(sonar_sensor_task_entry, (const portCHAR *)"Sonar
Sensor Task", 1024, NULL,
                           tskIDLE_PRIORITY + PRIO_SONAR_SENSOR_TASK,
&h_Task) != pdTRUE)
    {
        return (1);
    }

    if((xTimerStart(periodic_getDistance_timer, 0)) != pdTRUE)
    {
//        DEBUG_ERROR(1);
```

```
        return 1;
    }

    setSonar_sensorTaskHandle(h_Task);

    /* Return the createtask ret value */
    return 0;
}
/*
 * spi.c
 *
 *  Created on: Dec 1, 2017
 *      Author: Gunj Manseta
 */

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

#include "inc/hw_memmap.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"

#include "driverlib/gpio.h"
#include "driverlib/ssi.h"

#include "spi.h"

const SPI_Type SPI[4] = {SSI0_BASE, SSI1_BASE, SSI2_BASE, SSI3_BASE};

const SPI_SYSCTL_Type SPI_SYSCTL[4] = {SYSCTL_PERIPH_SSI0,
SYSCTL_PERIPH_SSI1, SYSCTL_PERIPH_SSI2, SYSCTL_PERIPH_SSI3};


void SPI_GPIO_init(SPI_t spi)
{

    if(spi==SPI_0)
    {
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
          GPIOPinConfigure(GPIO_PA2_SSI0CLK);
        GPIOPinConfigure(GPIO_PA3_SSI0FSS);
        GPIOPinConfigure(GPIO_PA4_SSI0XDAT0);
        GPIOPinConfigure(GPIO_PA5_SSI0XDAT1);

        // The pins are assigned as follows:
        //      PA5 - SSI0Tx
        //      PA4 - SSI0Rx
        //      PA3 - SSI0Fss
        //      PA2 - SSI0CLK
        GPIOPinTypeSSI(GPIO_PORTA_BASE, GPIO_PIN_5 | GPIO_PIN_4 |
GPIO_PIN_3 |
                       GPIO_PIN_2);
      }
    else if(spi==SPI_1)
    {
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
```

```
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
        GPIOPinConfigure(GPIO_PB5_SSI1CLK);
        GPIOPinConfigure(GPIO_PB4_SSI1FSS);
        GPIOPinConfigure(GPIO_PE4_SSI1XDAT0);
        GPIOPinConfigure(GPIO_PE5_SSI1XDAT1);

        // The pins are assigned as follows:
        //      PE4 - SSI0Tx
        //      PE5 - SSI0Rx
        //      PB4 - SSI0Fss
        //      PB5 - SSI0CLK
        GPIOPinTypeSSI(GPIO_PORTB_BASE, GPIO_PIN_5 | GPIO_PIN_4);
        GPIOPinTypeSSI(GPIO_PORTE_BASE, GPIO_PIN_5 | GPIO_PIN_4);
    }
    else if(spi==SPI_2)
    {
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
        GPIOPinConfigure(GPIO_PD3_SSI2CLK);
        GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_2);
        //GPIOPinConfigure(GPIO_PD2_SSI2FSS);
        GPIOPinConfigure(GPIO_PD1_SSI2XDAT0);
        GPIOPinConfigure(GPIO_PD0_SSI2XDAT1);

        // The pins are assigned as follows:
        //      PD1 - SSI0Tx
        //      PD0 - SSI0Rx
        //      PD2 - SSI0Fss
        //      PD3 - SSI0CLK
        GPIOPinTypeSSI(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 |
GPIO_PIN_3);
    }
}


void SPI_write_packet(SPI_t spi, uint8_t* p, size_t length)
{
      uint8_t i=0;
      while (i<length)
      {
            SPI_write_byte(spi, *(p+i));
            ++i;
      }
}

void SPI_read_packet(SPI_t spi, uint8_t* p, size_t length)
{
      uint8_t i=0;
      while (i<length)
      {
          SPI_write_byte(spi,0xFF);
            *(p+i) = SPI_read_byte(spi);
            ++i;
      }
}

void SPI0_IRQHandler()
{
```

```
}
/*
 * camera_interface.c
 *
 *  Created on: 29-Apr-2018
 *      Author: Gunj Manseta
 */
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "inc/hw_memmap.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/pin_map.h"


#include "my_uart.h"
#include "delay.h"
#include "spi.h"
#include "ov2640_regs.h"
#include "cam_header.h"
#include "camera_interface.h"

#define SLAVE_ADDRESS (0x60)

uint8_t read_fifo_burst()
{
    uint8_t temp = 0, temp_last = 0;
    uint32_t length = 0;
    length = read_fifo_length();
    uint8_t is_header = 0;

    if (length >= MAX_FIFO_SIZE) //512 kb
    {
        printf("ACK CMD Oversize");
        return 0;
    }
    if (length == 0 ) //0 kb
    {
        printf("ACK CMD Size is 0");
        return 0;
    }
    CS_LOW();

//    set_fifo_burst();//Set fifo burst mode
    transfer(BURST_FIFO_READ);
    temp =  transfer(0x00);
    length--;
    while ( length-- )
    {
        temp_last = temp;
        temp =  transfer(0x00);
        if (is_header == 1)
        {
            write(temp);
```

```
        }
        else
        if ((temp == 0xD8) & (temp_last == 0xFF))
        {
            is_header = 1;
            //printACKIMG();
            write(temp_last);
            write(temp);
        }
        if ( (temp == 0xD9) && (temp_last == 0xFF) ) //If find the end
,break while,
        {
            break;
        }
        DelayUs(15);
    }

    CS_HIGH();

    write_reg(ARDUCHIP_FIFO, FIFO_CLEAR_MASK);

    return length;
}

uint32_t SendFrame()
{
    //    myCAM.flush_fifo();
    write_reg(ARDUCHIP_FIFO, FIFO_RDPTR_RST_MASK);
     //Clear the capture done flag
    //     myCAM.clear_fifo_flag();
    write_reg(ARDUCHIP_FIFO, FIFO_CLEAR_MASK);
     //Start capture
    //     myCAM.start_capture();
    write_reg(ARDUCHIP_FIFO, FIFO_START_MASK);

    while (!get_bit(ARDUCHIP_TRIG , CAP_DONE_MASK));

    write_reg(ARDUCHIP_FIFO, FIFO_CLEAR_MASK);

    uint8_t ret = read_fifo_burst();

    DelayMs(20);

    return ret;

}


void CameraInit()
{
    SPI_clock_init(SPI_2,g_sysClock);
    SPI_init(SPI_2);
    I2C_init();

    while (1) {
      //Check if the ArduCAM SPI bus is OK
      write_reg(ARDUCHIP_TEST1, 0x55);
      uint8_t temp = read_reg(ARDUCHIP_TEST1);
```

```
      if (temp != 0x55) {
        //Serial.println(F("SPI interface Error!"));
        DelayUs(1000); continue;
      } else {
//          printf("Camera SPI working. \n");
          break;
      }
    }

     uint8_t vid = 0, pid = 0;
     wrSensorReg8_8(0xff, 0x01);
     DelayUs(1000);
     while(1)
     {
         vid = rdSensorReg8_8(OV2640_CHIPID_HIGH);
         pid = rdSensorReg8_8(OV2640_CHIPID_LOW);
         if ((vid == 0x26 ) && ( pid == 0x42 ))
         {
             printf("Found OV2640 module!\n");
             break;
         }


    }

    //CAM INIT on jpeg
    wrSensorReg8_8(0xff, 0x01);
    wrSensorReg8_8(0x12, 0x80);
    DelayUs(1000);
    wrSensorRegs8_8(OV2640_JPEG_INIT);
    DelayUs(1000);
    wrSensorRegs8_8(OV2640_YUV422);
    DelayUs(1000);
    wrSensorRegs8_8(OV2640_JPEG);
    DelayUs(1000);
    wrSensorReg8_8(0xff, 0x01);
    wrSensorReg8_8(0x15, 0x00);
//    wrSensorRegs8_8(OV2640_160x120_JPEG);
//    wrSensorRegs8_8(OV2640_320x240_JPEG);
    wrSensorRegs8_8(OV2640_640x480_JPEG);
    DelayUs(1000);
    wrSensorReg8_8(0xff, 0x00);
    wrSensorReg8_8(0x44, 0x32);

}

void I2C_init()
{
    //enable GPIO peripheral that contains I2C 0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);

    //reset module
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);

    // Configure the pin muxing for I2C0 functions on port B2 and B3.
    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
```

```
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);

    // Select the I2C function for these pins.
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    // Enable and initialize the I2C0 master module.  Use the system
clock for
    // the I2C0 module.  The last parameter sets the I2C data transfer
rate.
    // If false the data rate is set to 100kbps and if true the data rate
will
    // be set to 400kbps.
    I2CMasterInitExpClk(I2C0_BASE, g_sysClock, true);

    I2CTxFIFOFlush(I2C0_BASE);
    I2CRxFIFOFlush(I2C0_BASE);
    //clear I2C FIFOs
    //HWREG(I2C0_BASE + I2C_O_FIFOCTL) = 80008000;

}




uint8_t get_bit(uint8_t addr, uint8_t bit)
{
  uint8_t temp;
  temp = read_reg(addr);
  temp = temp & bit;
  return temp;
}


uint32_t read_fifo_length()
{
    uint8_t len1,len2,len3;
    uint32_t length=0;
    len1 = read_reg(FIFO_SIZE1);
    len2 = read_reg(FIFO_SIZE2);
    len3 = read_reg(FIFO_SIZE3) & 0x7f;
    length = ((len3 << 16) | (len2 << 8) | len1) & 0x07fffff;
    return length;
}

uint8_t transfer(uint8_t val)
{
    SPI_write_byte(SPI_2, val);
    uint8_t value = SPI_read_byte(SPI_2);
    return value;
}

uint8_t read_reg(uint8_t reg)
{
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
    SPI_write_byte(SPI_2, reg);
    SPI_read_byte(SPI_2);
    SPI_write_byte(SPI_2, 0xFF);
```

```
    uint8_t value = SPI_read_byte(SPI_2);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
    return value;
}

void write_reg(uint8_t reg, uint8_t value)
{
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 0);
    SPI_write_byte(SPI_2, reg | 0x80);
    SPI_read_byte(SPI_2);
    SPI_write_byte(SPI_2, value);
    SPI_read_byte(SPI_2);
    GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, GPIO_PIN_2);
}

void wrSensorRegs8_8(uint8_t **array_reg_value_pair)
{
    uint8_t reg_addr = 0;
    uint8_t reg_val = 0;
    uint8_t *next = array_reg_value_pair;
    uint32_t count = 0;
    while ((reg_addr != 0xff) | (reg_val != 0xff))
    {
      reg_addr = *(next);
      reg_val = *(next+1);
      wrSensorReg8_8(reg_addr, reg_val);
      next+=2;
      count++;
    }
//    printf("ACK CMD Count %u, NEXT: %u REG: 0x%x, Dat: 0x%x\n", count,
next, reg_addr, reg_val);
}

uint8_t rdSensorReg8_8(uint8_t reg)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, (SLAVE_ADDRESS)>>1, false);
    //specify register to be read
    I2CMasterDataPut(I2C0_BASE, reg);

    //send control byte and register address byte to slave device
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);

    //wait for MCU to finish transaction
    while(I2CMasterBusy(I2C0_BASE));

    //specify that we are going to read from slave device
    I2CMasterSlaveAddrSet(I2C0_BASE, (SLAVE_ADDRESS| 0x01) >>1, true);

    //send control byte and read from the register we
    //specified
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_RECEIVE);

    //wait for MCU to finish transaction
    while(I2CMasterBusy(I2C0_BASE));

    //return data pulled from the specified register
    uint8_t val =  I2CMasterDataGet(I2C0_BASE);
```

```
    while(I2CMasterBusy(I2C0_BASE));
    return val;
}


void wrSensorReg8_8(uint8_t reg, uint8_t data)
{
    I2CMasterSlaveAddrSet(I2C0_BASE, (SLAVE_ADDRESS>>1), false);
    I2CMasterDataPut(I2C0_BASE, reg);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_START);
    while(I2CMasterBusy(I2C0_BASE));

    I2CMasterDataPut(I2C0_BASE, data);
    //send next data that was just placed into FIFO
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_SEND_FINISH);
    // Wait until MCU is done transferring.
    while(I2CMasterBusy(I2C0_BASE));
}
/*
 * heartbeat.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"

#include "FreeRTOS.h"
#include "timers.h"

#include <comm_sender_task.h>
#include "delay.h"
#include "my_uart.h"

#define NUM_OF_TIMERS   2
static TimerHandle_t timer_handles[NUM_OF_TIMERS];

void vTimerCallback(TimerHandle_t h_timer)
{
    static uint32_t led_val = (GPIO_PIN_1 | GPIO_PIN_0);
    static uint32_t count = 0;
    if(h_timer == timer_handles[0])
    {
        if(count%5 == 0)
        {
            //Notify the comm_sender task with Heartbeat event
            //TODO:Check for return value
            NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_HEARTBEAT);
        }
//        if(count%30 == 0)
//        {
//            NOTIFY_COMM_OBJECT(EVENT_COMM_SENDER_BOARD_TYPE);
//        }
```

```
        count++;
    }
    //TIMER_LED_HEARTBEAT
    else if(h_timer == timer_handles[1])
    {
        led_val ^= (GPIO_PIN_1 | GPIO_PIN_0);
        GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_1 | GPIO_PIN_0, led_val);
    }
}

void heartbeat_start(uint32_t log_heartbeat_time_ms, uint32_t
led_heartbeat_time_ms)
{
    MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    MAP_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1 | GPIO_PIN_0);

    timer_handles[0] = xTimerCreate("TIMER_LOG_HEARTBEAT",
pdMS_TO_TICKS(log_heartbeat_time_ms) , pdTRUE,  (void*)0,
vTimerCallback);
    DEBUG_ERROR(timer_handles[0] == NULL);

    timer_handles[1] = xTimerCreate("TIMER_LED_HEARTBEAT",
pdMS_TO_TICKS(led_heartbeat_time_ms) , pdTRUE,  (void*)0,
vTimerCallback);

    DEBUG_ERROR(timer_handles[1] == NULL);

    if((xTimerStart(timer_handles[0], 0)) != pdTRUE)
    {
        DEBUG_ERROR(1);
    }

    if((xTimerStart(timer_handles[1], 0)) != pdTRUE)
    {
        DEBUG_ERROR(1);
    }
}
/**
* @file - nordic_driver.c
* @brief - Implementation file for the driver functions of the NRF240L
*
* @author Gunj/Ashish University of Colorado Boulder
* @date - 8 Dec 2017
**/


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

#include "driverlib/rom_map.h"
#include "driverlib/interrupt.h"
#include "my_uart.h"

#include "nordic_driver.h"
#include "spi.h"

//Commands Byte
```

```
#define NORDIC_TXFIFO_FLUSH_CMD    (0xE1)
#define NORDIC_RXFIFO_FLUSH_CMD    (0xE2)
#define NORDIC_W_TXPAYLD_CMD (0xA0)
#define NORDIC_R_RXPAYLD_CMD (0x61)
#define NORDIC_ACTIVATE_CMD        (0x50)
#define NORDIC_ACTIVATE_DATA (0x73)
#define NORDIC_RXPAYLD_W_CMD (0x60)
#define NORDIC_NOP                        (0xFF)
//Register Addresses
#define NORDIC_CONFIG_REG          (0x00)
#define NORDIC_EN_AA_REG         (0x01)
#define NORDIC_EN_RXADDR_REG     (0x02)
#define NODIC_SETUP_RETR_REG     (0x04)
#define NORDIC_STATUS_REG          (0x07)
#define NORDIC_RF_SETUP_REG        (0x06)
#define NORDIC_RF_CH_REG           (0x05)
#define NORDIC_TX_ADDR_REG         (0x10)
#define NORDIC_TX_ADDR_LEN         (5)

#define NORDIC_RX_ADDR_P0_REG    (0x0A)
#define NORDIC_RX_ADDR_P1_REG    (0x0B)
#define NORDIC_RX_ADDR_P2_REG    (0x0C)
#define NORDIC_RX_ADDR_P3_REG    (0x0D)
#define NORDIC_RX_ADDR_P4_REG    (0x0E)
#define NORDIC_RX_ADDR_P5_REG    (0x0F)

#define NORDIC_FIFO_STATUS_REG     (0x17)
#define NORDIC_RX_PW_P0_REG        (0x11)

#define DEFAULT_TX_ADDRESS_1B      (0xE7)
#define DEFAULT_TX_ADDRESS_2B      (0xE7)
#define DEFAULT_TX_ADDRESS_3B      (0xE7)
#define DEFAULT_TX_ADDRESS_4B      (0xE7)
#define DEFAULT_TX_ADDRESS_5B      (0xE7)

//Masks
#define NORDIC_CONFIG_MAX_RT_MASK       4
#define NORDIC_CONFIG_MAX_RT_INT(x)
((((uint8_t)x)<<NORDIC_CONFIG_MAX_RT_MASK)&(1<<NORDIC_CONFIG_MAX_RT_MASK)
)

#define NORDIC_CONFIG_RX_DR_MASK        6
#define NORDIC_CONFIG_RX_DR_INT(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_RX_DR_MASK)&(1<<NORDIC_CONFIG_RX_DR_M
ASK))

#define NORDIC_CONFIG_TX_DS_MASK        5
#define NORDIC_CONFIG_TX_DS_INT(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_TX_DS_MASK)&(1<<NORDIC_CONFIG_TX_DS_M
ASK))

#define NORDIC_CONFIG_PWR_UP_MASK       1
#define NORDIC_CONFIG_PWR_UP(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_PWR_UP_MASK)&(1<<NORDIC_CONFIG_PWR_UP
_MASK))

#define NORDIC_CONFIG_PRIM_RX_MASK      0
```

```
#define NORDIC_CONFIG_PRIM_RX(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_PRIM_RX_MASK)&(1<<NORDIC_CONFIG_PRIM_
RX_MASK))


#define NORDIC_STATUS_TX_FULL_MASK            (1<<0)
#define NORDIC_FIFO_STATUS_TX_FULL_MASK       (1<<5)
#define NORDIC_FIFO_STATUS_RX_FULL_MASK       (1<<1)
#define NORDIC_FIFO_STATUS_TX_EMPTY_MASK      (1<<4)
#define NORDIC_FIFO_STATUS_RX_EMPTY_MASK      (0<<5)

#define NORDIC_INT_MAXRT_MASK   (1<<3)
#define NORDIC_INT_TXDS_MASK    (1<<4)
#define NORDIC_INT_TXDR_MASK    (1<<5)

volatile uint8_t txconfigured = 0;
volatile uint8_t rxconfigured = 0;

volatile uint8_t transmitted = 0;
volatile uint8_t received = 0;
volatile uint8_t retry_error = 0;

static uint8_t using_interrupt = 0;

extern uint32_t g_sysClock;

void NRF_IntHandler(void);

static NRF_INT_HANDLER_T user_handler;

void NRF_gpioInit()
{
    //Enabling the GPIO PC4 for Nordic CE pin
    MAP_SysCtlPeripheralEnable(NORDIC_CE_SYSCTL_PORT);
    GPIOPinTypeGPIOOutput(NORDIC_CE_PORT, NORDIC_CE_PIN);
    GPIOPinWrite(NORDIC_CE_PORT, NORDIC_CE_PIN, 0);

    //Enabling the GPIO PC5 for Nordic CSN pin
    MAP_SysCtlPeripheralEnable(NORDIC_CSN_SYSCTL_PORT);
    GPIOPinTypeGPIOOutput(NORDIC_CSN_PORT, NORDIC_CSN_PIN);
    GPIOPinWrite(NORDIC_CSN_PORT, NORDIC_CSN_PIN, NORDIC_CSN_PIN);

    //Enabling the GPIO PC6 for Nordic IRQ pin
    MAP_SysCtlPeripheralEnable(NORDIC_IRQ_SYSCTL_PORT);

    GPIOIntDisable(NORDIC_IRQ_PORT,0xFFFF);
    GPIOPinTypeGPIOInput(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN);
    GPIOIntUnregister(NORDIC_IRQ_PORT);
    GPIOIntClear(NORDIC_IRQ_PORT,0xFFFF);
    GPIOIntTypeSet(NORDIC_IRQ_PORT, NORDIC_IRQ_PIN, GPIO_LOW_LEVEL);
//    GPIOIntRegister(NORDIC_IRQ_PORT, NRF_IntHandler);
//    GPIOIntDisable(NORDIC_IRQ_PORT,0xFFFF);
//    GPIOIntEnable(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN);
}

int8_t NRF_moduleInit(uint8_t use_interrupt, NRF_INT_HANDLER_T handler)
{
    SPI_clock_init(SPI_1, g_sysClock);
```

```
    SPI_init(SPI_1);
    DelayMs(1);

    NRF_gpioInit();
    if(use_interrupt)
    {
        using_interrupt = 1;
        user_handler = handler;
        GPIOIntRegister(NORDIC_IRQ_PORT, NRF_IntHandler);
        GPIOIntDisable(NORDIC_IRQ_PORT,0xFFFF);
        GPIOIntEnable(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN);
    }
    else
    {
        using_interrupt = 0;
    }
    return 0;
}

void NRF_moduleSetup(NRF_DataRate_t DR, NRF_Power_t power)
{
    //Clearing all interrupts
    NRF_write_status(0);
    //Disabling all interrupts and init in power down TX mode
    NRF_write_config(0x78);
    NRF_write_rf_ch(44);
    NRF_write_rf_setup((power<<1) | (DR<<3) | 1);
    //ADDR LEN as 5bytes
    NRF_write_register(0x03, 0x03);
    DelayMs(1);
}

void NRF_moduleDisable()
{
    using_interrupt = 0;
    uint8_t config = NRF_read_config();
    NRF_write_config(config & ~NORDIC_CONFIG_PWR_UP(1));
      SPI_disable(SPI_1);
      GPIOIntClear(NORDIC_IRQ_PORT,NORDIC_IRQ_PIN);
      GPIOIntUnregister(NORDIC_IRQ_PORT);
}

uint8_t NRF_read_register(uint8_t regAdd)
{
      //SPI_clear_RXbuffer(SPI_1); //used to clear the previously value in
the RX FIFO
      uint8_t readValue = 0;

      //CSN High to low for new command
      NRF_chip_disable();
      NRF_chip_enable();

      SPI_write_byte(SPI_1,regAdd);
      SPI_read_byte(SPI_1);  //used to clear the previously value in the
RX FIFO
      SPI_write_byte(SPI_1,0xFF);
      readValue = SPI_read_byte(SPI_1);
```

```
	//Marking the end of transaction by CSN high
	NRF_chip_disable();

	return readValue;
}

void NRF_write_command(uint8_t command)
{
	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_1,command);
	//SPI_clear_RXbuffer(SPI_1); //used to clear the previously value in
the RX FIFO
	SPI_read_byte(SPI_1);

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

void NRF_write_register(uint8_t regAdd, uint8_t value)
{
	//SPI_clear_RXbuffer(SPI_1); //used to clear the previously value in
the RX FIFO

	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(SPI_1,regAdd | 0x20);
	SPI_read_byte(SPI_1);  //used to clear the previously value in the
RX FIFO
	SPI_write_byte(SPI_1,value);
	SPI_read_byte(SPI_1);  //used to clear the previously value in the
RX FIFO

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

void NRF_write_status(uint8_t statusValue)
{
	NRF_write_register(NORDIC_STATUS_REG, statusValue);
}

uint8_t NRF_read_status()
{
    uint8_t readValue = 0;

    //CSN High to low for new command
    NRF_chip_disable();
    NRF_chip_enable();

    SPI_write_byte(SPI_1,NORDIC_NOP);
    readValue = SPI_read_byte(SPI_1);   //used to clear the previously
value in the RX FIFO
```

```
    //Marking the end of transaction by CSN high
    NRF_chip_disable();

    return readValue;
}

void NRF_write_config(uint8_t configValue)
{
     NRF_write_register(NORDIC_CONFIG_REG, configValue);
}

uint8_t NRF_read_config()
{
     return NRF_read_register(NORDIC_CONFIG_REG);
}

uint8_t NRF_read_rf_setup()
{
     return NRF_read_register(NORDIC_RF_SETUP_REG);
}

void NRF_write_rf_setup(uint8_t rfSetupValue)
{
     NRF_write_register(NORDIC_RF_SETUP_REG, rfSetupValue);
}

uint8_t NRF_read_rf_ch()
{
     return NRF_read_register(NORDIC_RF_CH_REG);
}

void NRF_write_rf_ch(uint8_t channel)
{
     NRF_write_register(NORDIC_RF_CH_REG, channel);
}

void NRF_write_En_AA(uint8_t data)
{
    NRF_write_register(NORDIC_EN_AA_REG, data);
}

uint8_t NRF_read_En_AA()
{
    return NRF_read_register(NORDIC_EN_AA_REG);
}

void NRF_write_setup_retry(uint8_t data)
{
    NRF_write_register(NODIC_SETUP_RETR_REG, data);
}

uint8_t NRF_read_setup_retry()
{
    return NRF_read_register(NODIC_SETUP_RETR_REG);
}

void NRF_read_TX_ADDR(uint8_t *address)
{
```

```
      uint8_t i = 0;

      NRF_chip_disable();
      NRF_chip_enable();

      SPI_write_byte(SPI_1,NORDIC_TX_ADDR_REG);
      SPI_read_byte(SPI_1);  //used to clear the previously value in the
RX FIFO
      //SPI_read_byte(SPI_1);      //used to clear the previously value in
the RX FIFO
      while(i < NORDIC_TX_ADDR_LEN)
      {
            SPI_write_byte(SPI_1, 0xFF); //Dummy to get the data
            *(address+i) = SPI_read_byte(SPI_1);
            i++;
      }

      NRF_chip_disable();
}

void NRF_write_TX_ADDR(uint8_t * tx_addr)
{
      NRF_chip_disable();
      NRF_chip_enable();

      SPI_write_byte(SPI_1,NORDIC_TX_ADDR_REG | 0x20);
      SPI_read_byte(SPI_1);  //used to clear the previously value in the
RX FIFO
      SPI_write_packet(SPI_1,tx_addr,NORDIC_TX_ADDR_LEN);
      SPI_flushRXFIFO(SPI_1);

      NRF_chip_disable();
}

void NRF_read_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *address)
{
    if(pipe_num > 5)
        return;
//    uint8_t i = 0;

    NRF_chip_disable();
    NRF_chip_enable();

    SPI_write_byte(SPI_1,(NORDIC_RX_ADDR_P0_REG + pipe_num));
    SPI_read_byte(SPI_1);   //used to clear the previously value in the
RX FIFO
    //SPI_read_byte(SPI_1); //used to clear the previously value in the
RX FIFO
    size_t ADDR_LEN = NORDIC_TX_ADDR_LEN;
    pipe_num > 2 ? ADDR_LEN = 1: 0;
//    while(i < ADDR_LEN)
//    {
//        SPI_write_byte(SPI_1, 0xFF);    //Dummy to get the data
//        *(address+i) = SPI_read_byte(SPI_1);
//        i++;
//    }
    SPI_read_packet(SPI_1, address, ADDR_LEN);
```

```
    NRF_chip_disable();
}

void NRF_write_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *rx_addr)
{
    if(pipe_num > 5)
        return;

    NRF_chip_disable();
    NRF_chip_enable();

    SPI_write_byte(SPI_1,(NORDIC_RX_ADDR_P0_REG + pipe_num) | 0x20);
    SPI_read_byte(SPI_1);   //used to clear the previously value in the
RX FIFO
    size_t ADDR_LEN = NORDIC_TX_ADDR_LEN;
    pipe_num > 1 ? ADDR_LEN = 1: 0;
    SPI_write_packet(SPI_1,rx_addr,ADDR_LEN);
    SPI_flushRXFIFO(SPI_1);

    NRF_chip_disable();
}


uint8_t NRF_read_fifo_status()
{
      return NRF_read_register(NORDIC_FIFO_STATUS_REG);
}

void NRF_flush_tx_fifo()
{
      NRF_write_command(NORDIC_TXFIFO_FLUSH_CMD);
}

void NRF_flush_rx_fifo()
{
      NRF_write_command(NORDIC_RXFIFO_FLUSH_CMD);
}

void NRF_activate_cmd()
{
      NRF_write_register(NORDIC_ACTIVATE_CMD, NORDIC_ACTIVATE_DATA);
}

void NRF_enable_RX_PIPE(uint8_t rx_pipe_number)
{
    if(rx_pipe_number > 5)
        return;
    uint8_t ret = NRF_read_register(NORDIC_EN_RXADDR_REG);
    NRF_write_register(NORDIC_EN_RXADDR_REG, ret | (1<<rx_pipe_number));

}
void NRF_disable_RX_PIPE(uint8_t rx_pipe_number)
{
    if(rx_pipe_number > 5)
        return;
    uint8_t ret = NRF_read_register(NORDIC_EN_RXADDR_REG);
    NRF_write_register(NORDIC_EN_RXADDR_REG, ret &
(~(1<<rx_pipe_number)));
```

```
}

static void NRF_mode_configure(NRF_Mode_t mode, uint8_t rx_pipe_number,
uint8_t addr[5], uint8_t payload_size)
{
      if(mode < 2)
      {
          NRF_radio_disable();
            uint8_t configureRead = NRF_read_config();

            if(mode == NRF_Mode_TX)
            {
                  txconfigured = 1;
                  configureRead &= ~(NORDIC_CONFIG_TX_DS_INT(1));// |
NORDIC_CONFIG_MAX_RT_INT(1));
                  NRF_flush_tx_fifo();
              NRF_write_En_AA(0);
              NRF_write_setup_retry(0);
              NRF_write_TX_ADDR(addr);
              NRF_write_RX_PIPE_ADDR(rx_pipe_number, addr);
              NRF_enable_RX_PIPE(rx_pipe_number);
              NRF_write_register((NORDIC_RX_PW_P0_REG), payload_size);
              NRF_write_config(configureRead | NORDIC_CONFIG_PWR_UP(1));
              DelayMs(2);
            }
            else
            {
                  rxconfigured = 1;
                  configureRead |= NORDIC_CONFIG_PWR_UP(1) |
NORDIC_CONFIG_PRIM_RX(1);
                  configureRead &= ~(NORDIC_CONFIG_RX_DR_INT(1));
                  NRF_flush_rx_fifo();
                  NRF_enable_RX_PIPE(rx_pipe_number);
                  NRF_write_RX_PIPE_ADDR(rx_pipe_number, addr);
                  NRF_write_register((NORDIC_RX_PW_P0_REG +
rx_pipe_number), payload_size);
                  NRF_write_config(configureRead);
                  NRF_radio_enable();
            }

            DelayMs(2);
            printf("NORDIC Configured in %s mode\n", ((mode)?"RX
MODE":"TX MODE"));

      }
      else
      {
            printf("INVALID MODE\n");
      }
}

void NRF_openReadPipe(uint8_t rx_pipe_number, uint8_t rx_addr[5], uint8_t
payload_size)
{
    NRF_mode_configure(NRF_Mode_RX, rx_pipe_number, rx_addr,
payload_size);
}
```

```
void NRF_openWritePipe(uint8_t tx_addr[5])
{
    NRF_mode_configure(NRF_Mode_TX, 0, tx_addr, 5);
//    NRF_mode_configure(NRF_Mode_TX, 0, tx_addr, 32);
}

void NRF_closeWritePipe()
{
    txconfigured = 0;
    uint8_t configureRead = NRF_read_config();
    configureRead |= (NORDIC_CONFIG_TX_DS_INT(1)  |
NORDIC_CONFIG_MAX_RT_INT(1));
    NRF_write_config(configureRead);
    NRF_disable_RX_PIPE(0);
}

void NRF_closeReadPipe(uint8_t rx_pipe_number)
{
    NRF_radio_disable();
    rxconfigured = 0;
    uint8_t configureRead = NRF_read_config();
    configureRead |= NORDIC_CONFIG_RX_DR_INT(1);
    NRF_write_config(configureRead);
    NRF_disable_RX_PIPE(rx_pipe_number);
}

void NRF_write_TXPayload(uint8_t *data, uint8_t len)
{
    NRF_chip_disable();
    NRF_chip_enable();
    SPI_write_byte(SPI_1, NORDIC_W_TXPAYLD_CMD);
    SPI_read_byte(SPI_1); //used to clear the previously value in the RX
FIFO

    SPI_write_packet(SPI_1,data, len);  //loading the FIFO with data
before enabling the CE pin
    SPI_flushRXFIFO(SPI_1);
    NRF_chip_disable();
}

void NRF_TX_pulse()
{
    NRF_radio_enable();
    //Delay of min 10us
    DelayUs(20);
    NRF_radio_disable();
}

int8_t NRF_transmit_data(uint8_t *data, uint8_t len, uint8_t toRXMode)
{
      if(txconfigured)
      {
          uint8_t configureRead = NRF_read_config();
          configureRead &= ~NORDIC_CONFIG_PRIM_RX(1);
          NRF_write_config(configureRead);
          configureRead = NRF_read_config();
          DelayUs(130);
```

```
            NRF_radio_disable();

            NRF_write_TXPayload(data, len);

            NRF_TX_pulse();

            printf("Data written");

            if(using_interrupt)
            {
             while(transmitted == 0 && retry_error == 0); //wait till TX
data is transmitted from FIFO
             if(retry_error)
             {
                 retry_error = 0;
                 printf("Data Retry Error\n");
             }
             else
             {
                 transmitted = 0; printf("Data Transmitted\n");
             }
            }
            else
            {
             uint8_t status = 0;
             do
             {
                 status = NRF_read_status();
             }while(!((NORDIC_STATUS_TX_DS_MASK |
NORDIC_STATUS_MAX_RT_MASK) & status));
             NRF_write_status(NORDIC_STATUS_TX_DS_MASK |
NORDIC_STATUS_MAX_RT_MASK | NORDIC_STATUS_MAX_RT_MASK);
            }

            if(toRXMode)
            {
             configureRead &= ~(NORDIC_CONFIG_PRIM_RX(1));
             NRF_write_config(configureRead);
             NRF_flush_rx_fifo();
             NRF_radio_enable();
            }

      }
      else
      {
            printf("TX mode not configured");
      }
      return 0;
}

NRF_read_RXPayload(uint8_t *data, uint8_t len)
{
    NRF_chip_enable();

    SPI_write_byte(SPI_1, NORDIC_R_RXPAYLD_CMD);
    SPI_read_byte(SPI_1);   //used to clear the previously value in the
RX FIFO
    SPI_read_packet(SPI_1,data,len);
```

```
    SPI_flush(SPI_1);

    NRF_chip_disable();
}

int8_t NRF_read_data(uint8_t *data, uint8_t len)
{
      if(rxconfigured)
      {
          NRF_radio_enable();
          uint8_t val = NRF_read_fifo_status();
          val = NRF_read_config();
          //TODO: Check how to move forward with this? Call this function
after we know that the data is avail or check with the
          //Status reg if data is available
          if(using_interrupt)
          {
              while(received == 0) //wait till RX data in FIFO
              {
                  val = NRF_read_fifo_status();
              }
              received = 0;
          }
          else
          {
              uint8_t status = 0;
              do
              {
                  status = NRF_read_status();
              }while(!(NORDIC_STATUS_RX_DR_MASK & status));
          }

            printf("Data received");

            NRF_read_RXPayload(data, len);

            printf("Data read");
      }
      else
      {
            printf("RX mode not configured");
      }
      return 0;
}

//#define SELF_TEST
#ifdef SELF_TEST

void Nordic_Test()
{
    NRF_moduleInit();
    NRF_moduleSetup(NRF_DR_1Mbps, NRF_PW_LOW);
    DelayMs(100);

    printf( "SPI Initialized\n");
    printf("Nordic Initialized\n");
    printf("Nordic Test\n");
//    NRF_write_status(0);
```

```
//    uint8_t sendValue = 0x08;
//    uint8_t readValue = 0;
//    NRF_write_config(sendValue);
//    readValue = NRF_read_config();
//    printf("Recv: 0x%x\n",readValue);
//    if(readValue == sendValue)
//    {
//        printf("Write/Read Config Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    DelayMs(5);
//
//    NRF_write_register(NORDIC_STATUS_REG,0);
//    sendValue = 44;
//    NRF_write_rf_ch(sendValue);
//    readValue = NRF_read_rf_ch();
//    if(readValue == sendValue)
//    {
//        printf("Write/Read RF CH Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    //sendValue = 0x0F;
//    sendValue = 0x07 ;
//    NRF_write_rf_setup(sendValue);
//    readValue = NRF_read_rf_setup();
//    if(readValue == sendValue)
//    {
//        printf("Write/Read RF Setup Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    NRF_write_register(0x03, 3);
//
////    uint8_t sendAddr[5] = {0xBA,0x56,0xBA,0x56,0xBA};
//    uint8_t sendAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
//    printf("TX ADDRESSES SET:
0x%x%x%x%x%x\n",sendAddr[0],sendAddr[1],sendAddr[2],sendAddr[3],sendAddr[
4]);
//    NRF_write_TX_ADDR(sendAddr);
//    uint8_t readAddr[5];
//    NRF_read_TX_ADDR(readAddr);
//    printf("TX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);
//
//    NRF_read_RX_P0_ADDR(readAddr);
//    printf("RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);
//
//    NRF_write_RX_P0_ADDR(sendAddr);
//    NRF_read_RX_P0_ADDR(readAddr);
```

```
//    printf("RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);




//    NRF_Mode_t mode = NRF_Mode_RX;
//    printf("Configuring NRF in %d mode",mode);
//    NRF_mode_configure(mode);
//    uint8_t Data[2] = {0};
//    NRF_read_data(Data,2);
//    printf("Nordic Data Recvd: 0x%x, 0x%x", Data[0],Data[1]);

    uint8_t sendAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
    NRF_openWritePipe(sendAddr);
    printf("Configuring NRF in TX mode");
    uint8_t readAddr[5];
    NRF_read_TX_ADDR(readAddr);
    logger_log(INFO,"TX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    //NRF_read_RX_P0_ADDR(readAddr);
    logger_log(INFO,"RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    NRF_read_RX_PIPE_ADDR(0, readAddr);
    logger_log(INFO,"RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    uint8_t Data[5] = {0x55,0xBB,0xBB,0xBB,0xBB};
    NRF_transmit_data(Data,5, false);
    printf("Nordic Data Sent: 0x%x, 0x%x", Data[0],Data[1]);

    printf("Nordic Test End\n");

    NRF_moduleDisable();
}
#endif

void NRF_IntHandler(void)
{
    MAP_IntMasterDisable();
        uint32_t int_status = GPIOIntStatus(NORDIC_IRQ_PORT, false);
        if(int_status & NORDIC_IRQ_PIN)
        {
            GPIOIntClear(NORDIC_IRQ_PORT, NORDIC_IRQ_PIN);
            uint8_t NRF_int_reason = NRF_read_status();
            if(NRF_int_reason & NORDIC_STATUS_TX_DS_MASK)
            {
                NRF_write_status(NRF_int_reason |
NORDIC_STATUS_TX_DS_MASK);
                transmitted = 1;
//                printf("NRF TX Complete\n");
            }
```

```
            if(NRF_int_reason & NORDIC_STATUS_RX_DR_MASK)
            {
                NRF_write_status(NRF_int_reason |
NORDIC_STATUS_RX_DR_MASK);
                NRF_flush_rx_fifo();
                //TODO: Notification to the handler for the Nordic Data
recv task
                user_handler();
                received = 1;
                printf("NRF RX Complete\n");
            }
            if(NRF_int_reason & NORDIC_STATUS_MAX_RT_MASK)
            {
                NRF_write_status(NRF_int_reason |
NORDIC_STATUS_MAX_RT_MASK);
                NRF_flush_tx_fifo();
                //TODO: Notification to the handler for the Nordic Data
recv task
                user_handler();
                retry_error = 1;
//                printf("NRF TX RETRY ERROR\n");
            }
        }
        MAP_IntMasterEnable();
}



/*
 * comm_sender_task.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdint.h>
#include <stdbool.h>
#include <limits.h>

#include "FreeRTOS.h"
#include "task.h"
#include "priorities.h"
#include "timers.h"

#include "my_uart.h"

#include "communication_interface.h"
#include "comm_sender_task.h"
#include "camera_interface.h"

#define COMM_SENDER_QUEUE_ITEMSIZE   (sizeof(COMM_MSG_T))
#define COMM_SENDER_QUEUE_LENGTH     20

extern const char * const BOARD_TYPE;
extern const char * const OS;
extern const char * const CODE_VERSION;
extern const char * const UID;
```

```
volatile uint8_t comm_senderTaskInitDone = 0;
static QueueHandle_t h_comm_senderQueue;
static TaskHandle_t h_comm_senderTask;

QueueHandle_t Comm_senderQueueHandle(QueueHandle_t handle, bool get)
{
    if(get)
        return h_comm_senderQueue;
    else
    {
        h_comm_senderQueue = handle;
        return h_comm_senderQueue;
    }
}

TaskHandle_t Comm_senderTaskHandle(TaskHandle_t handle, bool get)
{
    if(get)
        return h_comm_senderTask;
    else
    {
        h_comm_senderTask = handle;
        return h_comm_senderTask;
    }
}

static void comm_sender_task_entry(void *params)
{
    const TickType_t xMaxBlockTime = pdMS_TO_TICKS(5000);
    BaseType_t xResult;
    uint32_t notifiedValue = 0;
    while(1)
    {
        /* Wait to be notified of an interrupt. */
        xResult = xTaskNotifyWait( pdFALSE,    /* Don't clear bits on
entry. */
                           ULONG_MAX,        /* Clear all bits on exit.
*/
                           &notifiedValue, /* Stores the notified value.
*/
                           portMAX_DELAY);

        if( xResult == pdPASS )
        {
            /* A notification was received.  See which bits were set. */
            if(notifiedValue & EVENT_COMM_SENDER_BOARD_TYPE)
            {

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_COMM_MODULE,
BBG_LOGGER_MODULE);
                comm_msg.msg_id = MSG_ID_CLIENT_INFO_BOARD_TYPE;
                comm_msg.src_id = TIVA_COMM_MODULE;
                comm_msg.data.nothing = 1;
                strncpy(comm_msg.message,BOARD_TYPE,
sizeof(comm_msg.message));
                FILL_CHECKSUM(&comm_msg);
                COMM_SEND(&comm_msg);
                printf("BOARD_TYPE: %s\n",BOARD_TYPE);
```

```
            }

            if(notifiedValue & EVENT_COMM_SENDER_CODE_VERSION)
            {

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_COMM_MODULE,
BBG_LOGGER_MODULE);
                comm_msg.msg_id = MSG_ID_CLIENT_INFO_CODE_VERSION;
                comm_msg.src_id = TIVA_COMM_MODULE;
                comm_msg.data.nothing = 1;
                strncpy(comm_msg.message,CODE_VERSION,
sizeof(comm_msg.message));
                FILL_CHECKSUM(&comm_msg);
                COMM_SEND(&comm_msg);
                printf("CODE_VERSION: %s\n",CODE_VERSION);
            }

            if(notifiedValue & EVENT_COMM_SENDER_UID)
            {

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_COMM_MODULE,
BBG_LOGGER_MODULE);
                comm_msg.msg_id = MSG_ID_CLIENT_INFO_UID;
                comm_msg.src_id = TIVA_COMM_MODULE;
                comm_msg.data.nothing = 1;
                strncpy(comm_msg.message,UID, sizeof(comm_msg.message));
                FILL_CHECKSUM(&comm_msg);
                COMM_SEND(&comm_msg);
                printf("UID: %s\n",UID);
            }

            if(notifiedValue & EVENT_COMM_SENDER_HEARTBEAT)
            {
                static uint32_t count = 0;

COMM_CREATE_OBJECT(comm_msg,MY_TIVA_BOARD_ID,TIVA_HEART_BEAT_MODULE,
BBG_LOGGER_MODULE);
                comm_msg.msg_id = MSG_ID_HEARTBEAT;
                comm_msg.src_id = TIVA_COMM_MODULE;
                comm_msg.data.nothing = 1;
                strncpy(comm_msg.message,"HEARTBEAT",
sizeof(comm_msg.message));
                FILL_CHECKSUM(&comm_msg);
                COMM_SEND(&comm_msg);
                printf("[%u]HEARTBEAT\n",count++);
            }

            if(notifiedValue & EVENT_COMM_SENDER_STATUS)
            {
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("STATUS: %s\n",comm_msg.message);
                }
                else
```

```
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }

            if(notifiedValue & EVENT_COMM_SENDER_INFO)
            {
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("INFO: %s\n",comm_msg.message);
                }
                else
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }


            if(notifiedValue & EVENT_COMM_SENDER_MSG)
            {
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
                    comm_msg.msg_id = MSG_ID_MSG;
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("MSG: %s\n",comm_msg.message);
                }
                else
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }

            if(notifiedValue & EVENT_COMM_SENDER_ERROR)
            {
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
                    comm_msg.msg_id = MSG_ID_ERROR;
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("STATUS: %s\n",comm_msg.message);
                }
                else
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }

            if(notifiedValue & EVENT_COMM_SENDER_OBJECT_DETECTED)
            {
```

```
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
//                    comm_msg.msg_id = MSG_ID_OBJECT_DETECTED;
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("OBJECT DETECTED: %f
cm\n",comm_msg.data.distance_cm);
                    DelayUs(500);
                    //Sending the camera frame from here
                    SendFrame();
                }
                else
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }

            if(notifiedValue & EVENT_COMM_SENDER_PICTURE)
            {
                COMM_MSG_T comm_msg;
                if(h_comm_senderQueue &&
xQueueReceive(h_comm_senderQueue,&comm_msg,xMaxBlockTime))
                {
                    comm_msg.msg_id = MSG_ID_PICTURE;
                    FILL_CHECKSUM(&comm_msg);
                    COMM_SEND(&comm_msg);
                    printf("SENDING PICTURE of size:
%u\n",comm_msg.data.camera_packet->length);
                    //Now extract the CAMERA_PACKET and send the raw
frame from buffer pointer provided in the camera packet
                    //The camera frame data buffer should exit in the
memory
                    CAMERA_PACKET_T *packet;
                    if(comm_msg.data.camera_packet)
                    {
                        packet = comm_msg.data.camera_packet;
                    }
                    else
                    {
                        printf("[ERROR] NULL CAMERA PACKT");
                        packet->length = 0;
                        packet->frame = NULL;
                    }
                    COMM_SENDRAW(packet->frame,packet->length);
                    printf("SUCCESS: SEND PICTURE\n");
                }
                else
                {
                    printf("[Error] Q RECV %s\n",__FUNCTION__);
                }
            }

        }
//        else
//        {
//            printf("COMM_SENDER NOTIFICATION: TIMEOUT\n");
```

```
//        }
    }
}

uint8_t CommSenderTask_init()
{
    /* Creating a Queue required for Logging the msg */
    QueueHandle_t h_comm_senderQ = xQueueCreate(COMM_SENDER_QUEUE_LENGTH,
COMM_SENDER_QUEUE_ITEMSIZE);
    setComm_senderQueueHandle(h_comm_senderQ);

    TaskHandle_t h_comm_senderTask;

    xCOMM_SENDER_NOTIFY_MUTEX =  xSemaphoreCreateMutex();
    if(xCOMM_SENDER_NOTIFY_MUTEX == NULL)
    {
        printf("Semaphore Create Error. %s\n",__FUNCTION__);
    }

    /*Initializing the communication interface*/
    COMM_INIT();
//    uint8_t data[32] = {0};
//    NRF_read_data(data, 32);
    /* Create the task*/
    if(xTaskCreate(comm_sender_task_entry, (const portCHAR *)"Comm_sender
Task", 512, NULL,
                        tskIDLE_PRIORITY + PRIO_COMM_SENDERTASK,
&h_comm_senderTask) != pdTRUE)
    {
        return (1);
    }

    //Setting the comm_sender task handle for future use
    setComm_senderTaskHandle(h_comm_senderTask);
    /* Return the createtask ret value */
    return 0;
}




/*
 * sonar_sensor.c
 *
 *  Created on: 28-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "inc/hw_memmap.h"
#include "inc/hw_timer.h"
#include "inc/hw_types.h"
#include "driverlib/timer.h"
#include "driverlib/gpio.h"
```

```
#include "driverlib/interrupt.h"

#include "delay.h"
#include "sonar_sensor.h"
#ifdef DEBUG
#include "my_uart.h"
#endif

#ifdef DEBUG
#undef DEBUG
#endif

#define ULTRASONIC_PORT         GPIO_PORTK_BASE
#define ULTRASONIC_TRIGGER_PIN  GPIO_PIN_0
#define ULTRASONIC_ECHO_PIN     GPIO_PIN_1
#define HIGH(PIN)               PIN
#define LOW(PIN)                0

extern uint32_t g_sysClock;

//const float distance_factor = (1.0/120.0);
const float distance_factor = (1.0/16.0);
//static float distance_factor = (float)(1.0/(g_sysClock/1000000));

//start and end for echo pulse
volatile uint32_t pulse_down=0, pulse_up =0;

volatile uint8_t sensor_busy = 0;

void TimerInit();
void echo_interrupt();

void Sonar_sensor_init()
{
    //Configuring the timer required to capture the pulse
    TimerInit();

    //Configure Trigger pin
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    DelayUs(100);
    GPIOPinTypeGPIOOutput(ULTRASONIC_PORT, ULTRASONIC_TRIGGER_PIN);

    //Configure Echo pin
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    DelayUs(10);
    GPIOPinTypeGPIOInput(ULTRASONIC_PORT, ULTRASONIC_ECHO_PIN);
    GPIOIntEnable(ULTRASONIC_PORT, ULTRASONIC_ECHO_PIN);
    GPIOIntTypeSet(ULTRASONIC_PORT, ULTRASONIC_ECHO_PIN,GPIO_BOTH_EDGES);
    GPIOIntRegister(ULTRASONIC_PORT,echo_interrupt);
//    IntMasterEnable();

}

float sonarSensor_getDistance()
{
    uint32_t iteration = 0, retryCount = 0;
    float distance_old = 0, distance = 0;
    while(iteration < 10 && retryCount < 10)
```

```
    {
        //Check if the sensor is busy
        if(sensor_busy != 1)
        {
            //Give the required pulse of 10uS to trigger
            GPIOPinWrite(ULTRASONIC_PORT, ULTRASONIC_TRIGGER_PIN,
ULTRASONIC_TRIGGER_PIN);
            DelayUs(10);
            GPIOPinWrite(ULTRASONIC_PORT, ULTRASONIC_TRIGGER_PIN, 0);

            /*Wait while the reading is measured. The sensor_busy is
cleared by the falling edge of echo pin which
             *is done in the interrupt
             */
            while(sensor_busy != 0);

            //Converts
            pulse_up = pulse_down - pulse_up;
            distance =(float)(distance_factor * pulse_up);
            distance = distance/58;

    #ifdef DEBUG
            printf("[IN]Distance = %f cm \n" ,distance);
    #endif
            (distance_old < distance) ? distance_old = distance : 0;
            iteration++;
        }
        else
        {
            retryCount++;
    #ifdef DEBUG
            printf("Retry: %d\n" ,retryCount);
    #endif
        }
    }

    return (distance > distance_old) ?  distance : distance_old;
}

void TimerInit()
{
  SysCtlPeripheralEnable(SYSCTL_PERIPH_TIMER2);
  DelayUs(10);
  TimerConfigure(TIMER2_BASE, TIMER_CFG_PERIODIC_UP);
  //TimerEnable(TIMER2_BASE,TIMER_A);
}

void echo_interrupt()
{
    IntMasterDisable();
    //Clear interrupt flag
    GPIOIntClear(ULTRASONIC_PORT, ULTRASONIC_ECHO_PIN);

    /*Echo pulse rising edge*/
    if(GPIOPinRead(ULTRASONIC_PORT, ULTRASONIC_ECHO_PIN) ==
ULTRASONIC_ECHO_PIN)
    {
```

```
        HWREG(TIMER2_BASE + TIMER_O_TAV) = 0; //Loads value 0 into the
timer.
//        pulse_up = TimerValueGet(TIMER2_BASE,TIMER_A);
        pulse_up = 0;
        TimerEnable(TIMER2_BASE,TIMER_A);
        sensor_busy=1;
    }
    /*Echo pulse falling edge*/
    else
    {
        pulse_down = TimerValueGet(TIMER2_BASE,TIMER_A);
        TimerDisable(TIMER2_BASE,TIMER_A);
        sensor_busy=0;
    }

    IntMasterEnable();
}
/*
 * my_uart.c
 *
 *  Created on: 05-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdio.h>
#include <stdarg.h>
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_memmap.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/rom_map.h"
#include "driverlib/pin_map.h"
#include "my_uart.h"

#define UART_CONFIG_NORMAL  (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE
|UART_CONFIG_PAR_NONE)

char* convert(UART_T uart, unsigned int num, int base);

extern uint32_t g_sysClock;

const uint32_t UART[4] = {UART0_BASE, UART1_BASE, UART2_BASE,
UART3_BASE};

void UART_config(UART_T uart, BAUD_RATE_ENUM baudrate)
{

    if(uart == UART_0)
    {
        // Enable the GPIO Peripheral used by the UART.
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        // Enable UART0
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);
        while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_UART0))
        {
        }
```

```
        // Configure GPIO Pins for UART mode.
        MAP_GPIOPinConfigure(GPIO_PA0_U0RX);
        MAP_GPIOPinConfigure(GPIO_PA1_U0TX);
        MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    }
    else if(uart == UART_3)
    {
        // Enable the GPIO Peripheral used by the UART.
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        // Enable UART0
        MAP_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART3);
        while(!MAP_SysCtlPeripheralReady(SYSCTL_PERIPH_UART3))
        {
        }

        // Configure GPIO Pins for UART mode.
        MAP_GPIOPinConfigure(GPIO_PA4_U3RX);
        MAP_GPIOPinConfigure(GPIO_PA5_U3TX);
        MAP_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_4 | GPIO_PIN_5);


    }

#ifdef __USE_FREERTOS
    g_pUARTMutex[uart] = xSemaphoreCreateMutex();
#endif
    // Use the system clock oscillator as the UART clock source.
    //UARTClockSourceSet(UART[uart], UART_CLOCK_SYSTEM);
    UARTConfigSetExpClk(UART[uart], g_sysClock, baudrate,
UART_CONFIG_NORMAL);
    UARTEnable(UART[uart]);
}

void UART_putRAW(UART_T uart, const uint8_t *data, size_t len)
{
    while(UARTBusy(UART[uart]));
    while(len--)
    {
        UARTCharPut(UART[uart], *data++);
    }
}

size_t UART_getRAW(UART_T uart, uint8_t *data, size_t len)
{
    if(!UARTCharsAvail(UART[uart]))
        return 0;
    size_t i = 0, retrycount = 0;
    while(i<len && retrycount < 64)
    {
        int32_t c = UARTCharGetNonBlocking(UART[uart]);
        if(c != -1)
        {
            *(data+i) = c;
            retrycount = 0;
            i++;
        }
        else
```

```
            retrycount++;
    }
    return i;
}

void UART_putstr(UART_T uart, const char *str)
{
    while(*str)
    {
        if(*str == '\n')
            UARTCharPut(UART[uart], '\r');
        UARTCharPut(UART[uart], *str++);
    }
}

void UART_printf(UART_T uart, const char *fmt, ...)
{
    const char *p;
    int i;
    unsigned int u;
    char *s;
    double d;
    char str[10];
    va_list argp;

    va_start(argp, fmt);

    p=fmt;
    for(p=fmt; *p!='\0';p++)
    {
        if(*p != '%')
        {
            UART_putchar(uart,*p);
            continue;
        }

        p++;

        switch(*p)
        {
        case 'f' :
            d=va_arg(argp,double);
            if(d<0)
            {
                d=-d;
                UART_putchar(uart,'-');
            }
            snprintf(str,sizeof(str),"%.02f",d);
            UART_putstr(uart, str);
            break;
        case 'c' :
            i=va_arg(argp,int);
            UART0_putchar(i);
            break;
        case 'd' :
            i=va_arg(argp,int);
            if(i<0)
            {
```

```
                i=-i;
                UART0_putchar('-');
            }
            UART_putstr(uart, convert(uart,i,10));
            break;
        case 'o':
            i=va_arg(argp,unsigned int);
            UART_putstr(uart, convert(uart,i,8));
            break;
        case 's':
            s=va_arg(argp,char *);
            UART_putstr(uart, s);
            break;
        case 'u':
            u=va_arg(argp,unsigned int);
            UART_putstr(uart, convert(uart,u,10));
            break;
        case 'x':
            u=va_arg(argp,unsigned int);
            UART_putstr(uart, convert(uart,u,16));
            break;
        case '%':
            UART_putchar(uart,'%');
            break;
        }
    }

    va_end(argp);
}

static char buf0[35];
static char buf1[2];
static char buf2[2];
static char buf3[35];
char * const buff_arr[4] = {buf0, buf1, buf2, buf3};

char* convert(UART_T uart, unsigned int num, int base)
{
    //static char buf[50];
    char *ptr = buff_arr[uart];

    ptr=&(buff_arr[uart])[sizeof(buff_arr[uart])-1];
    *ptr='\0';
    do
    {
        *--ptr="0123456789abcdef"[num%base];
        num/=base;
    }while(num!=0);
    return(ptr);
}
/*
 * communication_interface.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#if 1
```

```
#include "communication_interface.h"

/* NRF COMM FUNCTIONS*/
void my_NRF_IntHandler()
{
}

volatile uint8_t count = 0;
int8_t comm_init_NRF()
{
    if(count)
    {
        count++;
        return 0;
    }
    int8_t status = NRF_moduleInit(NRF_USE_INTERRUPT, my_NRF_IntHandler);
    if(status == -1)
        return status;
    NRF_moduleSetup(NRF_DR_1Mbps, NRF_PW_MED);
    NRF_openReadPipe(1, RXAddr, sizeof(COMM_MSG_T)>32 ? 32 :
sizeof(COMM_MSG_T));
    NRF_openWritePipe(TXAddr);
    count++;

}

void comm_deinit_NRF()
{
    count--;
    if(count)
    {
        return;
    }
    NRF_closeReadPipe(1);
    NRF_closeWritePipe();
    NRF_moduleDisable();
}

int32_t comm_sendNRF_raw(uint8_t *data, uint32_t len)
{
    if(len <= 32)
    {
        NRF_transmit_data(data, len, true);
    }
//    else
//    {
//        size_t i = 0;
//        while(i < len)
//        {
//            NRF_transmit_data(data+i, 32 - (i%32), false);
//            i = i+32;
//        }
//    }

}

int32_t comm_recvNRF_raw(uint8_t *data, size_t len)
{
```

```
}
int32_t comm_recvNRF(COMM_MSG_T *p_comm_object)
{
    return NRF_read_data((uint8_t*)p_comm_object, sizeof(COMM_MSG_T));
}

#endif
/*
 * board_identification.c
 *
 *  Created on: 23-Apr-2018
 *      Author: Gunj Manseta
 */

#include <stdint.h>
#include <stdbool.h>

#define BOARD_UID_SHIFT 24
//TODO: Move the below constant strings to somewhere suitable
const char * const BOARD_TYPE = "TM4C1294XL";
const char * const OS = "FreeRTOS";
const char * const CODE_VERSION = "v1.0";
const char * const UID = "Gunj_Manseta";
/*
 * main.c
 *
 *  Created on: 05-Apr-2018
 *      Author: Gunj Manseta
 */

#include "application.h"

int main()
{
    application_run();

    //Will never come here
    return 0;
}
/*
 * comm_receiver_task.c
 *
 *  Created on: 26-Apr-2018
 *      Author: Gunj Manseta
 */

#include "priorities.h"

#include "my_uart.h"
#include "communication_object.h"
#include "communication_interface.h"
#include "comm_receiver_task.h"
#include "dispatcher_task.h"
#include "delay.h"

/* Create the entry task*/
static void comm_receiver_task_entry(void *params)
```

```
{
    const TickType_t xMaxBlockTime = pdMS_TO_TICKS(5000);
    /* Blocks on UART recv  OR get the notification from the UART RX
ISR*/
    /* Enqueues the recvd msg to the dispatcher task queue*/
    COMM_MSG_T recv_comm_msg;
    while(1)
    {
        memset(&recv_comm_msg, 0 , sizeof(recv_comm_msg));
        size_t ret = COMM_RECV(&recv_comm_msg);
        if(ret > 0)
        {
            if(ret != 32)
            {
                printf("RECV error. Data garbage\n");
            }else
            if(recv_comm_msg.dst_brd_id != MY_TIVA_BOARD_ID)
            {
                printf("Invalid Board Id\n");
            }
            else
            {
                /* Send to dispatcher */
                uint8_t ret =
ENQUEUE_NOTIFY_DISPATCHER_TASK(recv_comm_msg);
                if(ret == pdFAIL)
                {
                    printf("DISPATCHER NOTIFY ERROR\n");
                    continue;
                }
                printf("\n*******\n\
                SRCID:%u, SRC_BRDID:%u, DST_ID:%u, MSGID:%u\n\
                MSG:%s\n\
                Checksum:%u ?= %u\n********\n",\
                recv_comm_msg.src_id, recv_comm_msg.src_brd_id,
recv_comm_msg.dst_id,recv_comm_msg.msg_id,
recv_comm_msg.message,recv_comm_msg.checksum,
getCheckSum(&recv_comm_msg));
            }
        }
    }
}


/* Create the init */
uint8_t CommReceiverTask_init()
{
    TaskHandle_t h_comm_receiverTask;

    /*Initializing the communication interface. Not needed. Comm receiver
is doing it*/
    //COMM_INIT();
//    uint8_t data[32] = {0};
//    NRF_read_data(data, 32);
    /* Create the task*/
    if(xTaskCreate(comm_receiver_task_entry, (const portCHAR
*)"Comm_receiver Task", 128, NULL,
```

```
                        tskIDLE_PRIORITY + PRIO_COMM_RECEIVERTASK,
&h_comm_receiverTask) != pdTRUE)
    {
        return (1);
    }

    //Setting the comm_receiver task handle for future use
//    setComm_receiverTaskHandle(h_comm_receiverTask);
    /* Return the createtask ret value */
    return 0;
}

//*****************************************************************************
//
// startup_ccs.c - Startup code for use with TI's Code Composer Studio.
//
// Copyright (c) 2012-2017 Texas Instruments Incorporated.  All rights
reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware
Package.
//
//*****************************************************************************

#include <stdint.h>
#include "inc/hw_nvic.h"
#include "inc/hw_types.h"

//*****************************************************************************
//
// Forward declaration of the default fault handlers.
//
//*****************************************************************************
void ResetISR(void);
static void NmiSR(void);
static void FaultISR(void);
static void IntDefaultHandler(void);

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

```
//
// External declaration for the reset handler that is to be called when
the
// processor is started
//
//*****************************************************************************
******
extern void _c_int00(void);

//*****************************************************************************
******
//
// Linker variable that marks the top of the stack.
//
//*****************************************************************************
******
extern uint32_t __STACK_TOP;

//*****************************************************************************
******
//
// External declarations for the interrupt handlers used by the
application.
//
//*****************************************************************************
******
extern void xPortPendSVHandler(void);
extern void vPortSVCHandler(void);
extern void xPortSysTickHandler(void);

//*****************************************************************************
******
//
// The vector table.  Note that the proper constructs must be placed on
this to
// ensure that it ends up at physical address 0x0000.0000 or at the start
of
// the program if located at a start address other than 0.
//
//*****************************************************************************
******
#pragma DATA_SECTION(g_pfnVectors, ".intvecs")
void (* const g_pfnVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_TOP),
                                            // The initial stack pointer
    ResetISR,                               // The reset handler
    NmiSR,                                  // The NMI handler
    FaultISR,                               // The hard fault handler
    IntDefaultHandler,                      // The MPU fault handler
    IntDefaultHandler,                      // The bus fault handler
    IntDefaultHandler,                      // The usage fault handler
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    0,                                      // Reserved
    vPortSVCHandler,                        // SVCall handler
    IntDefaultHandler,                      // Debug monitor handler
```

```
    0,                                      // Reserved
    xPortPendSVHandler,                     // The PendSV handler
    xPortSysTickHandler,                    // The SysTick handler
    IntDefaultHandler,                      // GPIO Port A
    IntDefaultHandler,                      // GPIO Port B
    IntDefaultHandler,                      // GPIO Port C
    IntDefaultHandler,                      // GPIO Port D
    IntDefaultHandler,                      // GPIO Port E
    IntDefaultHandler,                      // UART0 Rx and Tx
    IntDefaultHandler,                      // UART1 Rx and Tx
    IntDefaultHandler,                      // SSI0 Rx and Tx
    IntDefaultHandler,                      // I2C0 Master and Slave
    IntDefaultHandler,                      // PWM Fault
    IntDefaultHandler,                      // PWM Generator 0
    IntDefaultHandler,                      // PWM Generator 1
    IntDefaultHandler,                      // PWM Generator 2
    IntDefaultHandler,                      // Quadrature Encoder 0
    IntDefaultHandler,                      // ADC Sequence 0
    IntDefaultHandler,                      // ADC Sequence 1
    IntDefaultHandler,                      // ADC Sequence 2
    IntDefaultHandler,                      // ADC Sequence 3
    IntDefaultHandler,                      // Watchdog timer
    IntDefaultHandler,                      // Timer 0 subtimer A
    IntDefaultHandler,                      // Timer 0 subtimer B
    IntDefaultHandler,                      // Timer 1 subtimer A
    IntDefaultHandler,                      // Timer 1 subtimer B
    IntDefaultHandler,                      // Timer 2 subtimer A
    IntDefaultHandler,                      // Timer 2 subtimer B
    IntDefaultHandler,                      // Analog Comparator 0
    IntDefaultHandler,                      // Analog Comparator 1
    IntDefaultHandler,                      // Analog Comparator 2
    IntDefaultHandler,                      // System Control (PLL, OSC,
BO)
    IntDefaultHandler,                      // FLASH Control
    IntDefaultHandler,                      // GPIO Port F
    IntDefaultHandler,                      // GPIO Port G
    IntDefaultHandler,                      // GPIO Port H
    IntDefaultHandler,                      // UART2 Rx and Tx
    IntDefaultHandler,                      // SSI1 Rx and Tx
    IntDefaultHandler,                      // Timer 3 subtimer A
    IntDefaultHandler,                      // Timer 3 subtimer B
    IntDefaultHandler,                      // I2C1 Master and Slave
    IntDefaultHandler,                      // Quadrature Encoder 1
    IntDefaultHandler,                      // CAN0
    IntDefaultHandler,                      // CAN1
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // Hibernate
    IntDefaultHandler,                      // USB0
    IntDefaultHandler,                      // PWM Generator 3
    IntDefaultHandler,                      // uDMA Software Transfer
    IntDefaultHandler,                      // uDMA Error
    IntDefaultHandler,                      // ADC1 Sequence 0
    IntDefaultHandler,                      // ADC1 Sequence 1
    IntDefaultHandler,                      // ADC1 Sequence 2
    IntDefaultHandler,                      // ADC1 Sequence 3
    0,                                      // Reserved
    0,                                      // Reserved
```

```
IntDefaultHandler,                      // GPIO Port J
IntDefaultHandler,                      // GPIO Port K
IntDefaultHandler,                      // GPIO Port L
IntDefaultHandler,                      // SSI2 Rx and Tx
IntDefaultHandler,                      // SSI3 Rx and Tx
IntDefaultHandler,                      // UART3 Rx and Tx
IntDefaultHandler,                      // UART4 Rx and Tx
IntDefaultHandler,                      // UART5 Rx and Tx
IntDefaultHandler,                      // UART6 Rx and Tx
IntDefaultHandler,                      // UART7 Rx and Tx
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // I2C2 Master and Slave
IntDefaultHandler,                      // I2C3 Master and Slave
IntDefaultHandler,                      // Timer 4 subtimer A
IntDefaultHandler,                      // Timer 4 subtimer B
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // Timer 5 subtimer A
IntDefaultHandler,                      // Timer 5 subtimer B
IntDefaultHandler,                      // Wide Timer 0 subtimer A
IntDefaultHandler,                      // Wide Timer 0 subtimer B
IntDefaultHandler,                      // Wide Timer 1 subtimer A
IntDefaultHandler,                      // Wide Timer 1 subtimer B
IntDefaultHandler,                      // Wide Timer 2 subtimer A
IntDefaultHandler,                      // Wide Timer 2 subtimer B
IntDefaultHandler,                      // Wide Timer 3 subtimer A
IntDefaultHandler,                      // Wide Timer 3 subtimer B
IntDefaultHandler,                      // Wide Timer 4 subtimer A
IntDefaultHandler,                      // Wide Timer 4 subtimer B
IntDefaultHandler,                      // Wide Timer 5 subtimer A
IntDefaultHandler,                      // Wide Timer 5 subtimer B
IntDefaultHandler,                      // FPU
0,                                      // Reserved
0,                                      // Reserved
IntDefaultHandler,                      // I2C4 Master and Slave
IntDefaultHandler,                      // I2C5 Master and Slave
IntDefaultHandler,                      // GPIO Port M
```

```
    IntDefaultHandler,                      // GPIO Port N
    IntDefaultHandler,                      // Quadrature Encoder 2
    0,                                      // Reserved
    0,                                      // Reserved
    IntDefaultHandler,                      // GPIO Port P (Summary or
P0)
    IntDefaultHandler,                      // GPIO Port P1
    IntDefaultHandler,                      // GPIO Port P2
    IntDefaultHandler,                      // GPIO Port P3
    IntDefaultHandler,                      // GPIO Port P4
    IntDefaultHandler,                      // GPIO Port P5
    IntDefaultHandler,                      // GPIO Port P6
    IntDefaultHandler,                      // GPIO Port P7
    IntDefaultHandler,                      // GPIO Port Q (Summary or
Q0)
    IntDefaultHandler,                      // GPIO Port Q1
    IntDefaultHandler,                      // GPIO Port Q2
    IntDefaultHandler,                      // GPIO Port Q3
    IntDefaultHandler,                      // GPIO Port Q4
    IntDefaultHandler,                      // GPIO Port Q5
    IntDefaultHandler,                      // GPIO Port Q6
    IntDefaultHandler,                      // GPIO Port Q7
    IntDefaultHandler,                      // GPIO Port R
    IntDefaultHandler,                      // GPIO Port S
    IntDefaultHandler,                      // PWM 1 Generator 0
    IntDefaultHandler,                      // PWM 1 Generator 1
    IntDefaultHandler,                      // PWM 1 Generator 2
    IntDefaultHandler,                      // PWM 1 Fault
};

//*****************************************************************************
******
//
// This is the code that gets called when the processor first starts
execution
// following a reset event.  Only the absolutely necessary set is
performed,
// after which the application supplied entry() routine is called.  Any
fancy
// actions (such as making decisions based on the reset cause register,
and
// resetting the bits in that register) are left solely in the hands of
the
// application.
//
//*****************************************************************************
******
void
ResetISR(void)
{
    //
    // Jump to the CCS C initialization routine.  This will enable the
    // floating-point unit as well, so that does not need to be done
here.
    //
    __asm("    .global _c_int00\n"
          "    b.w     _c_int00");
```

```
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a NMI.
This
// simply enters an infinite loop, preserving the system state for
examination
// by a debugger.
//
//*****************************************************************************
static void
NmiSR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives a fault
// interrupt.  This simply enters an infinite loop, preserving the system
state
// for examination by a debugger.
//
//*****************************************************************************
static void
FaultISR(void)
{
    //
    // Enter an infinite loop.
    //
    while(1)
    {
    }
}

//*****************************************************************************
//
// This is the code that gets called when the processor receives an
unexpected
// interrupt.  This simply enters an infinite loop, preserving the system
state
// for examination by a debugger.
//
//*****************************************************************************
static void
IntDefaultHandler(void)
```

```
{
    //
    // Go into an infinite loop.
    //
    while(1)
    {
    }
}
//*****************************************************************************
//
// priorities.h - Priorities for the various FreeRTOS tasks.
//
// Copyright (c) 2012-2017 Texas Instruments Incorporated.  All rights
reserved.
// Software License Agreement
//
// Texas Instruments (TI) is supplying this software for use solely and
// exclusively on TI's microcontroller products. The software is owned by
// TI and/or its suppliers, and is protected under applicable copyright
// laws. You may not combine this software with "viral" open-source
// software in order to form a larger program.
//
// THIS SOFTWARE IS PROVIDED "AS IS" AND WITH ALL FAULTS.
// NO WARRANTIES, WHETHER EXPRESS, IMPLIED OR STATUTORY, INCLUDING, BUT
// NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE. TI SHALL NOT, UNDER ANY
// CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL, OR CONSEQUENTIAL
// DAMAGES, FOR ANY REASON WHATSOEVER.
//
// This is part of revision 2.1.4.178 of the EK-TM4C123GXL Firmware
Package.
//
//*****************************************************************************

#ifndef __PRIORITIES_H__
#define __PRIORITIES_H__

//*****************************************************************************
//
// The priorities of the various tasks.
//
//*****************************************************************************
#define PRIO_COMM_SENDERTASK    1
#define PRIO_COMM_RECEIVERTASK  1
#define PRIO_DISPATCHERTASK     1
#define PRIO_SONAR_SENSOR_TASK  2

#endif // __PRIORITIES_H__
/*
    FreeRTOS V7.0.2 - Copyright (C) 2011 Real Time Engineers Ltd.
```

```
******************************************************************************
**
     *
*
     *    FreeRTOS tutorial books are available in pdf and paperback.
*
     *    Complete, revised, and edited pdf reference manuals are also
*
     *    available.
*
     *
*
     *    Purchasing FreeRTOS documentation will not only help you, by
*
     *    ensuring you get running as quickly as possible and with an
*
     *    in-depth knowledge of how to use FreeRTOS, it will also help
*
     *    the FreeRTOS project to continue with its mission of providing
*
     *    professional grade, cross platform, de facto standard solutions
*
     *    for microcontrollers - completely free of charge!
*
     *
*
     *    >>> See http://www.FreeRTOS.org/Documentation for details. <<<
*
     *
*
     *    Thank you for using FreeRTOS, and thank you for your support!
*
     *
*
******************************************************************************
**


    This file is part of the FreeRTOS distribution.

    FreeRTOS is free software; you can redistribute it and/or modify it
under
    the terms of the GNU General Public License (version 2) as published
by the
    Free Software Foundation AND MODIFIED BY the FreeRTOS exception.
    >>>NOTE<<< The modification to the GPL is included to allow you to
    distribute a combined work that includes FreeRTOS without being
obliged to
    provide the source code for proprietary components outside of the
FreeRTOS
    kernel.  FreeRTOS is distributed in the hope that it will be useful,
but
    WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY
    or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU General Public
License for
```

```
    more details. You should have received a copy of the GNU General
Public
    License and the FreeRTOS license exception along with FreeRTOS; if
not it
    can be viewed here: http://www.freertos.org/a00114.html and also
obtained
    by writing to Richard Barry, contact details for whom are available
on the
    FreeRTOS WEB site.

    1 tab == 4 spaces!

    http://www.FreeRTOS.org - Documentation, latest information, license
and
    contact details.

    http://www.SafeRTOS.com - A version that is certified for use in
safety
    critical systems.

    http://www.OpenRTOS.com - Commercial support, development, porting,
    licensing and training services.
*/

#ifndef FREERTOS_CONFIG_H
#define FREERTOS_CONFIG_H

/*-----------------------------------------------------------
 * Application specific definitions.
 *
 * These definitions should be adjusted for your particular hardware and
 * application requirements.
 *
 * THESE PARAMETERS ARE DESCRIBED WITHIN THE 'CONFIGURATION' SECTION OF
THE
 * FreeRTOS API DOCUMENTATION AVAILABLE ON THE FreeRTOS.org WEB SITE.
 *
 * See http://www.freertos.org/a00110.html.
 *----------------------------------------------------------*/

#define configUSE_PREEMPTION                1
#define configUSE_IDLE_HOOK                 0
#define configUSE_TICK_HOOK                 0
#define configUSE_MALLOC_FAILED_HOOK        1
//#define configCPU_CLOCK_HZ                  ( ( unsigned long )
50000000 )
#define configCPU_CLOCK_HZ                  ( ( unsigned long ) 16000000
)
#define configTICK_RATE_HZ                  ( ( portTickType ) 1000 )
#define configMINIMAL_STACK_SIZE            ( ( unsigned short ) 200 )
#define configTOTAL_HEAP_SIZE               ( ( size_t ) ( 30000 ) )
#define configMAX_TASK_NAME_LEN             ( 12 )
#define configUSE_TRACE_FACILITY            1
#define configUSE_16_BIT_TICKS              0
#define configIDLE_SHOULD_YIELD             0
#define configUSE_CO_ROUTINES               0
#define configUSE_MUTEXES                   1
#define configUSE_RECURSIVE_MUTEXES         1
```

```
#define configCHECK_FOR_STACK_OVERFLOW      2

#define configUSE_TASK_NOTIFICATIONS        1

/* Software timer related definitions. */
#define configUSE_TIMERS                        1
#define configTIMER_TASK_PRIORITY               3
#define configTIMER_QUEUE_LENGTH                10
#define configTIMER_TASK_STACK_DEPTH            configMINIMAL_STACK_SIZE

#define configMAX_PRIORITIES                16
#define configMAX_CO_ROUTINE_PRIORITIES     ( 2 )
#define configQUEUE_REGISTRY_SIZE           10

/* Set the following definitions to 1 to include the API function, or
zero
to exclude the API function. */

#define INCLUDE_vTaskPrioritySet            1
#define INCLUDE_uxTaskPriorityGet           1
#define INCLUDE_vTaskDelete                 1
#define INCLUDE_vTaskCleanUpResources       0
#define INCLUDE_vTaskSuspend                1
#define INCLUDE_vTaskDelayUntil             1
#define INCLUDE_vTaskDelay                  1
#define INCLUDE_uxTaskGetStackHighWaterMark 1

/* Be ENORMOUSLY careful if you want to modify these two values and make
sure
 * you read http://www.freertos.org/a00110.html#kernel_priority first!
 */
#define configKERNEL_INTERRUPT_PRIORITY         ( 7 << 5 )    /* Priority
7, or 0xE0 as only the top three bits are implemented.  This is the
lowest priority. */
#define configMAX_SYSCALL_INTERRUPT_PRIORITY     ( 5 << 5 )  /* Priority
5, or 0xA0 as only the top three bits are implemented. */

#endif /* FREERTOS_CONFIG_H */
/**
 * @brief
 *
 * @file socket_task.h
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#ifndef SOCKET_TASK_H
#define SOCKET_TASK_H

#include <signal.h>

sig_atomic_t socketTask_continue;

/**
 * @brief
 *
 * @param threadparam
 * @return void*
```

```
 */
void* socket_task_callback(void* threadparam);



#endif/*
 * delay.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef DELAY_H_
#define DELAY_H_

#include <stdint.h>
#include <unistd.h>

/** These functions does not guarentee the sleep time mentioned */

static inline void DelayS(uint32_t s)
{
    sleep(s);
}
static inline void DelayMs(uint32_t ms)
{
    usleep(ms*1000);
}

static inline void DelayUs(uint32_t us)
{
    usleep(us);
}


#endif /* DELAY_H_ */
/**
* @file - nordic_driver.h
* @brief - Header file for the driver functions of the NRF240L
*
* @author Gunj University of Colorado Boulder
* @date - 19th April 2017
**/


#ifndef __NORDIC_DRIVER_H__
#define __NORDIC_DRIVER_H__

#if 1

#include <stdbool.h>
#include <stdint.h>

#include "delay.h"

#include "spi.h"
#include "mraa/spi.h"
#include "mraa/gpio.h"
```

```
#define NORDIC_STATUS_RX_DR_MASK              (1<<6)
#define NORDIC_STATUS_TX_DS_MASK              (1<<5)
#define NORDIC_STATUS_MAX_RT_MASK            (1<<4)

typedef void (*NRF_INT_HANDLER_T)(void);

typedef enum{

      NRF_Mode_TX = 0,
      NRF_Mode_RX = 1

}NRF_Mode_t;

typedef enum{

    NRF_DR_1Mbps = 0,
    NRF_DR_2Mbps = 1

}NRF_DataRate_t;

typedef enum{

    NRF_PW_LOW = 0,
    NRF_PW_MED = 2,
    NRF_PW_HIGH = 3

}NRF_Power_t;

extern mraa_gpio_context NRF_CSN_GPIO;
extern mraa_gpio_context NRF_CE_GPIO;


/**
* @brief - Enable the chip select connection to Nordic
* @return void
**/
static inline void NRF_chip_enable()
{
      mraa_result_t status = mraa_gpio_write(NRF_CSN_GPIO, 0);
    if (status != MRAA_SUCCESS)
    {
    }
      DelayUs(50);
}

/**
* @brief - Disable the chip select connection to Nordic
* @return void
**/
static inline void NRF_chip_disable()
{
    mraa_result_t status = mraa_gpio_write(NRF_CSN_GPIO, 1);
    if (status != MRAA_SUCCESS)
    {
    }
}
```

```
/**
* @brief - Enable TX/RX from the Nordic module
* @return void
**/
static inline void NRF_radio_enable()
{
    mraa_result_t status = mraa_gpio_write(NRF_CE_GPIO, 1);
    if (status != MRAA_SUCCESS)
    {
    }
}

/**
* @brief - Disable TX/RX from the Nordic module
* @return void
**/
static inline void NRF_radio_disable()
{
    mraa_result_t status = mraa_gpio_write(NRF_CE_GPIO, 0);
    if (status != MRAA_SUCCESS)
    {
    }
}

/**
* @brief - Initialize the NRF module
* Initialized the GPIO connections pertaining to the Nordic module
* @return int8_t
**/
int8_t NRF_moduleInit(uint8_t use_interrupt, NRF_INT_HANDLER_T handler);

/**
* @brief - Disable the GPIO connections set up earlier for the Nordic
module
* @return void
**/
void NRF_moduleDisable();

/**
* @brief - Read a register from the NRF module
* @param - regAdd uint8_t
* @return uint8_t
**/
uint8_t NRF_read_register(uint8_t regAdd);

/**
* @brief - Write to a register from the NRF module
* @param - regAdd uint8_t
* @param - value uint8_t
* @return void
**/
void NRF_write_register(uint8_t regAdd, uint8_t value);

/**
* @brief - Write to the NRF module's status register
* @param - statusValue uint8_t
* @return void
```

```
**/
void NRF_write_status(uint8_t statusValue);

/**
* @brief - Read the NRF module's status register
* @return uint8_t
**/
uint8_t NRF_read_status();

/**
* @brief - Write to the NRF module's config register
* @param - configValue uint8_t
* @return void
**/
void NRF_write_config(uint8_t configValue);

/**
* @brief - Read the NRF module's config register
* @return uint8_t
**/
uint8_t NRF_read_config();


/**
* @brief - Read the NRF module's RF setup register
* @return uint8_t
**/
uint8_t NRF_read_rf_setup();

/**
* @brief - Write to the NRF module's RF setup register
* @param - rfStatusValue uint8_t
* @return void
**/
void NRF_write_rf_setup(uint8_t rfSetupValue);

/**
* @brief - Read the NRF module's RF CH register
* @return uint8_t
**/
uint8_t NRF_read_rf_ch();

/**
* @brief - Write to the NRF module's RF CH register
* @param - channel uint8_t
* @return void
**/
void NRF_write_rf_ch(uint8_t channel);

/**
* @brief - Reads 5 bytes of the NRF module's TX ADDR register
* @param - address uint8_t *
* @return void
**/
void NRF_read_TX_ADDR(uint8_t * address);

/**
* @brief - Writes 5 bytes of the NRF module's TX ADDR register
```

```
* @param - tx_addr uint8_t *
* @return void
**/
void NRF_write_TX_ADDR(uint8_t * tx_addr);

/**
* @brief - Read the NRF module's FIFO status register
* @return address uint8_t
**/
uint8_t NRF_read_fifo_status();

/**
* @brief - Send the command FLUSH_TX to the NRF module
* @return void
**/
void NRF_flush_tx_fifo();

/**
* @brief - Send the command FLUSH_RX to the NRF module
* @return void
**/
void NRF_flush_rx_fifo();

/**
* @brief - Send the activation command to the NRF module
* Activates the features: R_RX_PL_WID, W_ACK_PAYLOAD, W_TX_PAYLOAD_NOACK
* @return void
**/

void NRF_moduleSetup(NRF_DataRate_t DR, NRF_Power_t power);

void NRF_write_status(uint8_t statusValue);

uint8_t NRF_read_status();

void NRF_activate_cmd();

void NRF_read_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *address);
void NRF_write_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *rx_addr);

void NRF_write_En_AA(uint8_t data);
uint8_t NRF_read_En_AA();
void NRF_write_setup_retry(uint8_t data);
uint8_t NRF_read_setup_retry();

int32_t NRF_read_data(uint8_t *data, uint8_t len);
int32_t NRF_transmit_data(uint8_t *data, uint8_t len, uint8_t toRXMode);

void NRF_write_TXPayload(uint8_t *data, uint8_t len);
void NRF_TX_pulse();


void NRF_openReadPipe(uint8_t rx_pipe_number, uint8_t rx_addr[5], uint8_t
payload_size);

void NRF_openWritePipe(uint8_t tx_addr[5]);

void NRF_closeWritePipe();
```

```
void NRF_closeReadPipe(uint8_t rx_pipe_number);



#endif /* __NORDIC_DRIVER_H__ */
#endif
/*
 * communication_object.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef COMMUNICATION_OBJECT_H_
#define COMMUNICATION_OBJECT_H_

#include <string.h>

#ifndef BOARD_UID_SHIFT
#define BOARD_UID_SHIFT 24
#endif

#define GET_BOARD_UID_FROM_LOG_ID(id)    ((uint32_t)((id &
(0xFFU<<BOARD_UID_SHIFT))>>BOARD_UID_SHIFT))
#define GET_LOG_ID_FROM_LOG_ID(id)       ((id &
(~(0xFFU<<BOARD_UID_SHIFT))))


typedef enum
{
    MSG_ID_HEARTBEAT = 0,
    MSG_ID_MSG,
    MSG_ID_SENSOR_STATUS,
    MSG_ID_ERROR,
    MSG_ID_SENSOR_INFO,
    MSG_ID_INFO,
    MSG_ID_PICTURE,
    MSG_ID_OBJECT_DETECTED,
    MSG_ID_CLIENT_INFO_BOARD_TYPE,
    MSG_ID_CLIENT_INFO_UID,
    MSG_ID_CLIENT_INFO_CODE_VERSION,

    //The request id from the beaglebone
    MSG_ID_GET_SENSOR_STATUS,
    MSG_ID_GET_SENSOR_INFO,
    MSG_ID_GET_CLIENT_INFO_BOARD_TYPE,
    MSG_ID_GET_CLIENT_INFO_UID,
    MSG_ID_GET_CLIENT_INFO_CODE_VERSION,
    LAST_ID, //THIS ID IS JUST TO CALCULATE THE NUM OF IDS. THIS IS NOT
USED ANYWHERE This cannot be more than 255
}MSG_ID_T;

#define NUM_OF_ID   LAST_ID

const static char * const MSG_ID_STRING[NUM_OF_ID]  =
{
    "HEARTBEAT",
```

```
    "MSG",
    "STATUS",
    "ERROR",
    "INFO",
    "PICTURE",
    "OBJECT_DETECTED",
    "CLIENT_INFO_BOARD_TYPE",
    "CLIENT_INFO_UID",
    "CLIENT_INFO_CODE_VERSION",
    //The request id from the beaglebone
    "GET_SENSOR_STATUS",
    "GET_SENSOR_INFO",
    "GET_CLIENT_INFO_BOARD_TYPE",
    "GET_CLIENT_INFO_UID",
    "GET_CLIENT_INFO_CODE_VERSION",
};

//FOR DST and SRC Board ID
#define BBG_BOARD_ID        (0x00)
#define TIVA_BOARD1_ID      (0x01)
#define XYZ_TIVA_BOARD_ID   (0x02)

#define MY_TIVA_BOARD_ID    TIVA_BOARD1_ID

//For src and dst module ID
//Add all the modules' UID here for TIVA BOARD
#define TIVA_HEART_BEAT_MODULE   (1)
#define TIVA_SENSOR_MODULE       (2)
#define TIVA_CAMERA_MODULE       (3)
#define TIVA_COMM_MODULE         (4)

//Add all modules' UID here for BBG Board
#define BBG_LOGGER_MODULE       (1)
#define BBG_COMM_MODULE         (2)
#define BBG_SOCKET_MODULE       (3)
#define BBG_XYZ_MODULE          (4)

typedef uint8_t MSG_ID;
typedef uint8_t SRC_ID;
typedef uint8_t SRC_BOARD_ID;
typedef uint8_t DST_BOARD_ID;
typedef uint8_t DST_ID;

//This should be followed immediately by the PICTURE msg id
typedef struct cam_packet
{
    size_t length;
    void* frame;
}CAMERA_PACKET_T;

/*32byte LOG MESSAGE STRUCTURE*/
typedef struct COMM_MSG
{
    SRC_ID src_id;
    SRC_BOARD_ID src_brd_id;
    DST_ID dst_id;
    DST_BOARD_ID dst_brd_id;
    MSG_ID msg_id;
```

```
    union custom_data
    {
        float distance_cm;
        float sensor_value;
        CAMERA_PACKET_T *camera_packet;
        size_t nothing;
    }data;
    char message[18];
    uint16_t checksum;
}COMM_MSG_T;

static size_t COMM_MSG_SIZE = sizeof(COMM_MSG_T);

static uint16_t getCheckSum(const COMM_MSG_T *comm_msg)
{
    uint16_t checkSum = 0;
    uint8_t sizeOfPayload = sizeof(COMM_MSG_T) - sizeof(comm_msg-
>checksum);
    uint8_t *p_payload = (uint8_t*)comm_msg;
    int i;
    for(i = 0; i < sizeOfPayload; i++)
    {
        checkSum += *(p_payload+i);
    }
    return checkSum;
}

/*Return true if a match, return 0 is not a match*/
static inline uint8_t verifyCheckSum(const COMM_MSG_T *comm_msg)
{
    return (getCheckSum(comm_msg) == comm_msg->checksum);
}


#endif /* COMMUNICATION_OBJECT_H_ */
/*
 * communication_interface.h
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#ifndef COMMUNICATION_INTERFACE_H_
#define COMMUNICATION_INTERFACE_H_

#include <stdbool.h>
#include <stdint.h>

#include "my_uart.h"
#include "nordic_driver.h"
#include "communication_object.h"

#define NRF_USE_INTERRUPT   (1)
#define NRF_NOTUSE_INTERRUPT   (0)

//#define COMM_TYPE_NRF
//#define RUN_TIME_SWITCH
#ifdef RUN_TIME_SWITCH
```

```
volatile uint8_t comm_type_uart = 1;

#define COMM_INIT()                 comm_init_NRF();
comm_init_UART(BAUD_115200)
void COMM_SEND(COMM_MSG_T comm_object)
{
    if(comm_type_uart)
    {
        comm_sendUART(comm_object);
    }
    else
    {
        comm_sendNRF(comm_object);
    }
}
#else
#ifdef COMM_TYPE_NRF
#define COMM_INIT(fd)               comm_init_NRF()
#define COMM_DEINIT(fd)             comm_deinit_NRF()
#define COMM_SEND(p_comm_object)    comm_sendNRF(p_comm_object)
#define COMM_SENDRAW(packet,len)    comm_sendNRF_raw(packet, len)
#define COMM_RECV(p_comm_object)    comm_recvNRF(p_comm_object);
#else
#define COMM_INIT()                 comm_init_UART()
//Will be used only on BBG
#define COMM_DEINIT(fd)             comm_deinit_UART(fd)
#define COMM_SEND(p_comm_object)    comm_sendUART(p_comm_object)
#define COMM_SENDRAW(packet,len)    comm_sendUARTRAW(packet,len)
#define COMM_RECV(p_comm_object)    comm_recvUART(p_comm_object)
#endif
#endif
#define RX_PIPE 1

//0x54,0x4d,0x52,0x68,0x7C
static uint8_t TXAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
static uint8_t RXAddr[5] = {0xC2,0xC2,0xC2,0xC2,0xC2};

#ifdef TIVA_BOARD
static inline void comm_init_UART()
{
    UART3_config(BAUD_921600);
}

static inline void comm_deinit_UART(int fd){}


static inline void comm_sendUARTRAW(uint8_t* packet, size_t len)
{
    UART3_putRAW(packet,len);
}

static inline void comm_sendUART(COMM_MSG_T *p_comm_object)
{
    UART3_putRAW((uint8_t*)p_comm_object, sizeof(COMM_MSG_T));
    /* This is needed to mark the end of send as the receiving side needs
the line termination as the BeagleBone has opened the UART is canonical
mode*/
    //UART3_putchar('\n');
```

```
}

static inline size_t comm_recvUART(COMM_MSG_T *p_comm_object)
{
    size_t ret = UART3_getRAW((uint8_t*)p_comm_object,
sizeof(COMM_MSG_T));
    return ret;
}

#else
//For BBG

static inline UART_FD_T comm_init_UART()
{
    return UART_Open(COM_PORT4);
}

static inline void comm_deinit_UART(UART_FD_T fd)
{
    UART_Close(fd);
}

static inline int32_t comm_sendUART(COMM_MSG_T *p_comm_object)
{
    return UART_putRAW((void*)p_comm_object,sizeof(COMM_MSG_T));
}
static inline int32_t comm_sendUARTRAW(COMM_MSG_T * comm_object, size_t
len)
{
    return UART_putRAW((void*)comm_object,len);
}

static inline int32_t comm_recvUART(COMM_MSG_T *comm_object)
{
    int32_t available = UART_dataAvailable(100);
    if(available == 1)
    {
        return UART_read((void*)comm_object,sizeof(COMM_MSG_T));
    }
    else
        return available;
}

#endif

//For BBG end


int8_t comm_init_NRF();
void comm_deinit_NRF();

int32_t comm_sendNRF_raw(uint8_t *data, size_t len);

//TODO:
int32_t comm_recvNRF_raw(uint8_t *data, size_t len);
int32_t comm_recvNRF(COMM_MSG_T *p_comm_object);

static inline int32_t comm_sendNRF(COMM_MSG_T *p_comm_object)
```

```
{
    return NRF_transmit_data((uint8_t*)(p_comm_object),
sizeof(COMM_MSG_T), true);
}


#endif /* COMMUNICATION_INTERFACE_H_ */
/**
 * @brief
 *
 * @file posixTimer.h
 * @author Gunj Manseta
 * @date 2018-03-18
 */

#ifndef POSIX_TIMER_H
#define POSIX_TIMER_H


#include <time.h>
#include <linux/types.h>
#include <stdint.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>


#define MICROSEC (1000000)

/**
 * @brief REgsiter the timer handler
 *
 * @param timer_id
 * @param timer_handler
 * @param handlerArgs
 * @return int
 */
int register_timer(timer_t *timer_id, void (*timer_handler)(union
sigval), void *handlerArgs);


/**
 * @brief Starts the timer
 *
 * @param timer_id
 * @param time_usec
 * @param oneshot
 * @return int
 */
int start_timer(timer_t timer_id , uint64_t time_usec, uint8_t oneshot);

/**
 * @brief Stops the timer
 *
 * @param timer_id
 * @return int
 */
int stop_timer(timer_t timer_id);
```

```
/**
 * @brief Destroys the timer
 *
 * @param timer_id
 * @return int
 */
int delete_timer(timer_t timer_id);

#endif
/**
 * @brief
 *
 * @file sensor_common_object.h
 * @author Gunj Manseta
 * @date 2018-03-11
 */
#ifndef SENS_COMM_OBJ_H
#define SENS_COMM_OBJ_H

#include <stdint.h>
#include <semaphore.h>


/***
 * Required for remote client server communication
 **/
typedef enum
{
    GET_TEMP_C,
    GET_TEMP_F,
    GET_TEMP_K,
    GET_LUX,
    GET_DAY_NIGHT,
    GET_DISTANCE_CM,
    GET_DISTANCE_M,
    CONN_CLOSE_REQ,
    CONN_KILL_APP_REQ,
    CONN_CLOSE_RSP,
    CONN_KILL_APP_RSP,
    GET_FUNC,

}REMOTE_REQRSP_ID;

typedef struct
{
    REMOTE_REQRSP_ID request_id;

}REMOTE_REQUEST_T;

typedef struct
{
    REMOTE_REQRSP_ID rsp_id;
    union data{
        float floatingData;
        uint8_t isNight;
    }data;
    char metadata[20];
```

```
}REMOTE_RESPONSE_T;



typedef uint8_t*    P_BUFF_T;
typedef uint8_t     DEV_REG_T;
typedef size_t      BUFF_LEN_T;

typedef enum
{
    ASYNC = 0,
    SYNC  = 1
}SYNC_TYPE_T;

typedef struct
{
    SYNC_TYPE_T is_sync;
    sem_t       *sync_semaphore;
    DEV_REG_T   dev_addr;
    P_BUFF_T    *reg_value;
    BUFF_LEN_T  buffLen;

}OBJECT_PACKET_T;

#define SENSOR_MAKE_PACKET_SYNCTYPE(p_packet, p_sem) { p_packet->is_sync
= SYNC; if(NULL != p_sem){p_packet->sync_semaphore = p_sem;} }

#define SENSOR_MAKE_PACKET_ASYNCTYPE(p_packet)   {p_packet->is_sync =
ASYNC; p_packet->sync_semaphore = NULL;}

#define SENSOR_MAKE_PACKET_RW_1DATA(p_packet)   (p_packet->buffLen = 1)
#define SENSOR_MAKE_PACKET_RW_2DATA(p_packet)   (p_packet->buffLen = 2)

static inline int  SENSOR_FILL_OBJECT_DATA(OBJECT_PACKET_T *packet,
DEV_REG_T devaddr, P_BUFF_T *val, BUFF_LEN_T len)
{
    packet->dev_addr = devaddr;
    packet->reg_value = val;
    packet->buffLen = len;
}


#endif/**
 * @brief
 *
 * @file light_sensor_task.h
 * @author Gunj Manseta
 * @date 2018-03-11
 */

#ifndef LIGHTSENSOR_TASK_H
#define LIGHTSENSOR_TASK_H

#include <stdlib.h>
#include <errno.h>
#include <string.h>
```

```
#include <mqueue.h>
#include "common_helper.h"
#include "my_time.h"
#include "error_data.h"
#include "sensor_common_object.h"

//#define LightT_MSG_SIZE 40

//typedef char LIGHT_TASK_MSGDATA_T;


typedef enum
{
    DAY = 0,
    NIGHT = 1
}DAY_STATE_T;

typedef enum
{
    LIGHT_MSG_TASK_STATUS,
    LIGHT_MSG_TASK_GET_STATE,
    LIGHT_MSG_TASK_READ_DATA,
    LIGHT_MSG_TASK_WRITE_CMD,
    LIGHT_MSG_TASK_POWERDOWN,
    LIGHT_MSG_TASK_POWERUP,
    LIGHT_MSG_TASK_EXIT,
}LIGHTTASKQ_MSGID_T;


typedef struct
{
    LIGHTTASKQ_MSGID_T msgID;
    TASK_IDENTIFIER_T sourceID;
    OBJECT_PACKET_T packet;
    //LOGGER_TASK_MSGDATA_T msgData[LightT_MSG_SIZE];

}LIGHTTASKQ_MSG_T;


/**
 * @brief Defines a light struct with the name given and params with some
default values
 *
 */
#define DEFINE_LIGHT_STRUCT(name,msgId,sourceId) \
    LIGHTTASKQ_MSG_T name = {       \
        .msgID      = msgId,        \
        .sourceID   = sourceId,     \
        .packet     = {0}           \
    };

/**
 * @brief Get the Handle LightTaskQueue object
 *
 * @return mqd_t
 */
mqd_t getHandle_LightTaskQueue();
```

```
/**
 * @brief
 *
*/
#define POST_MESSAGE_LIGHTTASK(p_lightstruct)  \
    do{ \
        __POST_MESSAGE_LIGHTTASK(getHandle_LightTaskQueue(),
p_lightstruct, sizeof(*p_lightstruct),20); \
    }while(0)

/**
 * @brief
 *
 */
#define POST_MESSAGE_LIGHTTASK_EXIT(p_lightstruct)  \
    do{ \
        __POST_MESSAGE_LIGHTTASK(getHandle_LightTaskQueue(),
p_lightstruct, sizeof(*p_lightstruct),50); \
    }while(0)

/**
 * @brief
 *
 * @param queue
 * @param lightstruct
 * @param light_struct_size
 */
static inline void __POST_MESSAGE_LIGHTTASK(mqd_t queue, const
LIGHTTASKQ_MSG_T *lightstruct, size_t light_struct_size, int prio)
{
    if(-1 == mq_send(queue, (const char*)lightstruct, light_struct_size,
prio))
    {
        //LOG_STDOUT(ERROR "LIGHT:MQ_SEND:%s\n",strerror(errno));
        LOG_STDOUT(WARNING "LIGHT:MQ_SEND\n");
    }
}

/**
 * @brief Get the LightTask state object MT-safe
 *
 * @return DAY_STATE_T
 */
DAY_STATE_T getLightTask_state();

/**
 * @brief Get the LightTask lux object. MT-safe as it calls a MT-safe
function within
 *
 * @return float
 */
float getLightTask_lux();

/**
 * @brief Entry point of the light task thread
 *
 * @param threadparam
 * @return void*
```

```
 */
void* light_task_callback(void *threadparam);

#endif/**
 * @brief
 *
 * @file main_task.h
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#ifndef MAIN_TASK_H
#define MAIN_TASK_H

#include <mqueue.h>
#include <errno.h>
#include <string.h>
#include "common_helper.h"
#include "error_data.h"


#define MT_MSG_SIZE 20

typedef char MAINT_TASK_MSGDATA_T;

typedef enum
{
    MT_MSG_STATUS_RSP,
    MT_MSG_INIT_SUCCESS_RSP,

}MAINTASKQ_MSGID_T;

typedef struct
{
    MAINTASKQ_MSGID_T msgID;
    TASK_IDENTIFIER_T sourceID;
    MAINT_TASK_MSGDATA_T msgData[MT_MSG_SIZE];

}MAINTASKQ_MSG_T;


/**
 * @brief Defines a Main task queue struct with the name given and params
with some default values
 *
 */
#define DEFINE_MAINTASK_STRUCT(name,msgId,sourceId) \
    MAINTASKQ_MSG_T name = {      \
        .msgID      = msgId,        \
        .sourceID   = sourceId,     \
        .msgData    = {0}           \
    };

/**
 * @brief
 *
 */
#define POST_MESSAGE_MAINTASK(p_maintaskstruct, format, ...)  \
```

```
    do{ \
        snprintf((p_maintaskstruct)->msgData,sizeof((p_maintaskstruct)-
>msgData),format, ##__VA_ARGS__);   \
        __POST_MESSAGE_MAINTASK(getHandle_MainTaskQueue(),
p_maintaskstruct, sizeof(*p_maintaskstruct)); \
    }while(0)

/**
 * @brief Post message to the main task using the main task queue handle
and giving struct
 *
 * @param queue
 * @param main_task_struct
 * @param maintask_struct_size
 */
static inline void __POST_MESSAGE_MAINTASK(mqd_t queue, const
MAINTASKQ_MSG_T *main_task_struct, size_t maintask_struct_size)
{
    if(-1 == mq_send(queue, (const char*)main_task_struct,
maintask_struct_size, 20))
    {
        LOG_STDOUT(ERROR "MAIN:MQ_SEND:%s\n",strerror(errno));
    }
}

/**
 * @brief Get the Handle MainTaskQueue object
 *
 * @return mqd_t
 */
mqd_t getHandle_MainTaskQueue();

/**
 * @brief entry point for the main task
 *
 * @return int
 */
int main_task_entry();

#endif/**
 * @brief
 *
 * @file my_time.h
 * @author Gunj Manseta
 * @date 2018-03-18
 */
#ifndef TIME_H
#define TIME_H

/**
 * @brief Get the time string object
 *
 * @param timeString
 * @param len
 * @return int
 */
int get_time_string(char *timeString, int len);
```

```
#endif/**
 * @brief
 *
 * @file temperature_sensor_task.h
 * @author Gunj Manseta
 * @date 2018-03-11
 */

#ifndef TEMPSENSOR_TASK_H
#define TEMPSENSOR_TASK_H

#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <mqueue.h>
#include "common_helper.h"
#include "my_time.h"
#include "error_data.h"
#include "sensor_common_object.h"


typedef enum
{
    TEMP_MSG_TASK_STATUS,
    TEMP_MSG_TASK_GET_TEMP,
    TEMP_MSG_TASK_READ_DATA,
    TEMP_MSG_TASK_WRITE_CMD,
    TEMP_MSG_TASK_POWERDOWN,
    TEMP_MSG_TASK_POWERUP,
    TEMP_MSG_TASK_EXIT,

}TEMPERATURETASKQ_MSGID_T;


typedef struct
{
    TEMPERATURETASKQ_MSGID_T msgID;
    TASK_IDENTIFIER_T sourceID;
    OBJECT_PACKET_T packet;
    //LOGGER_TASK_MSGDATA_T msgData[LightT_MSG_SIZE];

}TEMPERATURETASKQ_MSG_T;


/**
 * @brief Defines a temp struct with the name given and params with some
default values
 *
 */
#define DEFINE_TEMP_STRUCT(name,msgId,sourceId) \
    TEMPERATURETASKQ_MSG_T name = {      \
        .msgID      = msgId,        \
        .sourceID   = sourceId,     \
        .packet     = {0}           \
    };

/**
```

```
 * @brief Get the Handle TemperatureTaskQueue object
 *
 * @return mqd_t
 */
mqd_t getHandle_TemperatureTaskQueue();

/**
 * @brief
 *
 */
#define POST_MESSAGE_TEMPERATURETASK(p_tempstruct)  \
    do{  \
        __POST_MESSAGE_TEMPERATURETASK(getHandle_TemperatureTaskQueue(),
p_tempstruct, sizeof(*p_tempstruct),20); \
    }while(0)

/**
 * @brief
 *
 */
#define POST_MESSAGE_TEMPERATURETASK_EXIT(p_tempstruct)  \
    do{  \
        __POST_MESSAGE_TEMPERATURETASK(getHandle_TemperatureTaskQueue(),
p_tempstruct, sizeof(*p_tempstruct),50); \
    }while(0)

/**
 * @brief
 *
 * @param queue
 * @param p_tempstruct
 * @param temp_struct_size
 */
static inline void __POST_MESSAGE_TEMPERATURETASK(mqd_t queue, const
TEMPERATURETASKQ_MSG_T *p_tempstruct, size_t temp_struct_size, int prio)
{
    if(-1 == mq_send(queue, (const char*)p_tempstruct, temp_struct_size,
prio))
    {
        //LOG_STDOUT(ERROR "TEMP:MQ_SEND:%s\n",strerror(errno));
        LOG_STDOUT(WARNING "TEMP:MQ_SEND\n");
    }
}

/**
 * @brief Get the TempTask temperature object MT-safe
 *
 * @return float
 */float getTempTask_temperature();

/**
 * @brief Entry point of the temp task thread
 *
 * @param threadparam
 * @return void*
 */
void* temperature_task_callback(void *threadparam);
```

```
#endif/**
 * @brief
 *
 * @file my_signals.h
 * @author Gunj Manseta
 * @date 2018-03-18
 */

#ifndef MY_SIGNALS_H
#define MY_SIGNALS_H

#include <sys/types.h>
#include <unistd.h>
#include <signal.h>
#include <stdint.h>

#define REG_SIG_USR1   (1<<0)
#define REG_SIG_USR2   (1<<1)
#define REG_SIG_INT          (1<<2)
#define REG_SIG_TERM   (1<<3)
#define REG_SIG_TSTP   (1<<4)
#define REG_SIG_ALL          (0x1F)

typedef uint8_t REG_SIGNAL_FLAG_t ;

/**
 * @brief Register asignal handler for specific signal masks
 *
 * @param sa
 * @param handler
 * @param signalMask
 * @return int
 */
int register_signalHandler(struct sigaction *sa, void (*handler)(int),
REG_SIGNAL_FLAG_t signalMask);



#endif
/**
 * @brief
 *
 * @file comm_sender_task.h
 * @author Gunj Manseta
 * @date 2018-04-26
 */

#ifndef COMM_SENDER_H
#define COMM_SENDER_H

#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <mqueue.h>

#include "common_helper.h"
```

```
#include "communication_object.h"

#define COMM_CREATE_OBJECT(name, src_board_id, source_id, dest_id)
COMM_MSG_T name = { .src_brd_id = src_board_id, .src_id = source_id,
.dst_id = dest_id, .dst_brd_id = TIVA_BOARD1_ID }
#define COMM_OBJECT_MSGID(comm_msg,msgid)   comm_msg.msg_id = msgid
#define COMM_DST_BOARD_ID(comm_msg,dst_board_id)    comm_msg.dst_brd_id =
dst_board_id
#define FILL_CHECKSUM(p_comm_msg)           do{ (p_comm_msg)->checksum =
getCheckSum(p_comm_msg); }while(0)
#define COMM_FILL_MSG(comm_msg,p_str)
strncpy(comm_msg.message,p_str,sizeof(comm_msg.message))

/**
 * @brief Get the Handle CommSenderTaskQueue object
 *
 * @return mqd_t
 */
mqd_t getHandle_CommSenderTaskQueue();


#define POST_MESSAGE_COMM_SENDTASK(p_comm_msg, format, ...)  \
    do{ \
        (strlen(format)>0) ? snprintf((p_comm_msg)-
>message,sizeof((p_comm_msg)->message),format, ##__VA_ARGS__): 0;   \
        FILL_CHECKSUM(p_comm_msg);   \
        __POST_MESSAGE_COMM_SENDTASK(getHandle_CommSenderTaskQueue(),
(p_comm_msg), sizeof(*p_comm_msg), 20); \
    }while(0)

#define POST_MESSAGE_COMM_SENDTASK_EXIT(format, ...)  \
    do{ \
        COMM_MSG_T comm_msg;    \
        (strlen(format)>0) ?
snprintf(comm_msg.message,sizeof(comm_msg.message),format,
##__VA_ARGS__):0; \
        COMM_OBJECT_MSGID(comm_msg,0xFF); \
        COMM_DST_BOARD_ID(comm_msg,BBG_BOARD_ID); \
        __POST_MESSAGE_COMM_SENDTASK(getHandle_CommSenderTaskQueue(),
&comm_msg, sizeof(comm_msg), 20); \
    }while(0)

/**
 * @brief
 *
 * @param queue
 * @param comm_msg
 * @param comm_msg_size
 * @param prio
 */
static inline void __POST_MESSAGE_COMM_SENDTASK(mqd_t queue, const
COMM_MSG_T *comm_msg, size_t comm_msg_size, int prio)
{
    if(-1 == mq_send(queue, (const char*)comm_msg, comm_msg_size, prio))
    {
        LOG_STDOUT(ERROR "COMM_SEND:MQ_SEND:%s\n",strerror(errno));
    }
}
```

```
/**
 * @brief
 *
 * @param board_id
 */
static inline void send_GET_CLIENT_INFO_CODE_VERSION(uint8_t board_id)
{
    COMM_CREATE_OBJECT(comm_msg,BBG_BOARD_ID, BBG_XYZ_MODULE,
TIVA_COMM_MODULE);
    COMM_OBJECT_MSGID(comm_msg,MSG_ID_CLIENT_INFO_CODE_VERSION);
    COMM_DST_BOARD_ID(comm_msg,board_id);
    FILL_CHECKSUM(&comm_msg);
    POST_MESSAGE_COMM_SENDTASK((&comm_msg), "BBG/Req/CodeV");
}

/**
 * @brief
 *
 * @param board_id
 */
static inline void send_GET_CLIENT_INFO_BOARD_TYPE(uint8_t board_id)
{
    COMM_CREATE_OBJECT(comm_msg,BBG_BOARD_ID, BBG_XYZ_MODULE,
TIVA_COMM_MODULE);
    COMM_OBJECT_MSGID(comm_msg,MSG_ID_CLIENT_INFO_BOARD_TYPE);
    COMM_DST_BOARD_ID(comm_msg,board_id);
    FILL_CHECKSUM(&comm_msg);
    POST_MESSAGE_COMM_SENDTASK(&comm_msg, "BBG/Req/BType");
}

/**
 * @brief
 *
 * @param board_id
 */
static inline void send_GET_CLIENT_INFO_UID(uint8_t board_id)
{
    COMM_CREATE_OBJECT(comm_msg,BBG_BOARD_ID, BBG_XYZ_MODULE,
TIVA_COMM_MODULE);
    COMM_OBJECT_MSGID(comm_msg,MSG_ID_GET_CLIENT_INFO_UID);
    COMM_DST_BOARD_ID(comm_msg,board_id);
    FILL_CHECKSUM(&comm_msg);
    POST_MESSAGE_COMM_SENDTASK(&comm_msg, "BBG/Req/UID");
}

/**
 * @brief
 *
 * @param board_id
 */
static inline void send_GET_DISTANCE(uint8_t board_id, uint8_t
src_module_id)
{
    COMM_CREATE_OBJECT(comm_msg,BBG_BOARD_ID, src_module_id,
TIVA_SENSOR_MODULE);
    COMM_OBJECT_MSGID(comm_msg,MSG_ID_GET_SENSOR_STATUS);
    COMM_DST_BOARD_ID(comm_msg,board_id);
```

```
    FILL_CHECKSUM(&comm_msg);
    POST_MESSAGE_COMM_SENDTASK(&comm_msg, "BBG/Req/Distance");
}


/**
 * @brief
 *
 * @param threadparam
 * @return void*
 */
void* comm_sender_task_callback(void *threadparam);

#endif
/**
 * @brief
 *
 * @file logger_task.h
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#ifndef LOGGER_TASK_H
#define LOGGER_TASK_H

#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <mqueue.h>
#include "common_helper.h"
#include "my_time.h"
#include "error_data.h"
#include "communication_object.h"


#define LT_MSG_SIZE 40

typedef char LOGGER_TASK_MSGDATA_T;

typedef enum
{
    LT_MSG_TASK_STATUS,
    LT_MSG_LOG,
    LT_MSG_TASK_EXIT,
    LT_MSG_COMM_MSG

}LOGGERTASKQ_MSGID_T;

typedef enum
{
    LOG_ERROR,
    LOG_WARNING,
    LOG_INFO,
    LOG_ALL

}LOG_LEVEL_T;
```

```
typedef struct
{
    LOGGERTASKQ_MSGID_T msgID;
    char timestamp[20];
    LOG_LEVEL_T loglevel;
    TASK_IDENTIFIER_T sourceID;
    union{
    LOGGER_TASK_MSGDATA_T msgData[LT_MSG_SIZE];
    COMM_MSG_T commMsg;
    }msgData;
}LOGGERTASKQ_MSG_T;

/**
 * @brief Defines a Log struct with the name given and params with some
default values
 *
 */
#define DEFINE_LOG_STRUCT(name,msgId,sourceId) \
    LOGGERTASKQ_MSG_T name = {      \
        .msgID      = msgId,        \
        .loglevel   = LOG_ALL,      \
        .sourceID   = sourceId,     \
        .timestamp  = {0},          \
        .msgData.msgData    = {0}           \
    };

#define LOG_FILL_COMM_MSG(p_log_struct, comm_msg) \
    memcpy(&(p_log_struct)->msgData.commMsg,&comm_msg,
sizeof((p_log_struct)->msgData.commMsg))

/**
 * @brief Set the Log loglevel
 *
 * @param log_msg
 * @param loglevel
 */
static inline void set_Log_loglevel(LOGGERTASKQ_MSG_T *log_msg,
LOG_LEVEL_T loglevel)
{
    log_msg->loglevel = loglevel;
}

/**
 * @brief Set the Log currentTimestamp to the currentTime as "sec.nsec"
 *
 * @param log_msg
 */
static inline void set_Log_currentTimestamp(LOGGERTASKQ_MSG_T *log_msg)
{
    get_time_string(log_msg->timestamp,sizeof(log_msg->timestamp));
}

/**
 * @brief Get the Handle LoggerTaskQueue object
 *
 * @return mqd_t
 */
mqd_t getHandle_LoggerTaskQueue();
```

```
/**
 * @brief
 *
 */
#define POST_COMM_MSG_LOGTASK(p_logstruct, comm_msg)  \
    do{ \
        (p_logstruct)->msgID = LT_MSG_COMM_MSG; \
        LOG_FILL_COMM_MSG(p_logstruct, comm_msg); \
        set_Log_currentTimestamp(p_logstruct); \
        __POST_MESSAGE_LOGTASK(getHandle_LoggerTaskQueue(), p_logstruct,
sizeof(*(p_logstruct)), 20); \
    }while(0)

/**
 * @brief
 *
 */
#define POST_MESSAGE_LOGTASK(p_logstruct, format, ...)  \
    do{ \
        snprintf((p_logstruct)->msgData.msgData,sizeof((p_logstruct)-
>msgData.msgData),format, ##__VA_ARGS__);   \
        set_Log_currentTimestamp(p_logstruct); \
        __POST_MESSAGE_LOGTASK(getHandle_LoggerTaskQueue(), p_logstruct,
sizeof(*p_logstruct), 20); \
    }while(0)

#define POST_MESSAGE_LOGTASK_EXIT(p_logstruct, format, ...)  \
    do{ \
        snprintf((p_logstruct)->msgData.msgData,sizeof((p_logstruct)-
>msgData.msgData),format, ##__VA_ARGS__);   \
        set_Log_currentTimestamp(p_logstruct); \
        __POST_MESSAGE_LOGTASK(getHandle_LoggerTaskQueue(), p_logstruct,
sizeof(*p_logstruct), 20); \
    }while(0)

/**
 * @brief Post message to the log using the log queue handle and giving
log struct
 *
 * @param queue
 * @param logstruct
 * @param log_struct_size
 */
static inline void __POST_MESSAGE_LOGTASK(mqd_t queue, const
LOGGERTASKQ_MSG_T *logstruct, size_t log_struct_size, int prio)
{
    if(-1 == mq_send(queue, (const char*)logstruct, log_struct_size,
prio))
    {
        LOG_STDOUT(ERROR "LOGGER:MQ_SEND:%s\n",strerror(errno));
    }
}

/**
 * @brief Entry point of the logger task thread
 *
 * @param threadparam
```

```
 * @return void*
 */
void* logger_task_callback(void *threadparam);

#endif/**
 * @brief
 *
 * @file comm_recv_task.h
 * @author Gunj Manseta
 * @date 2018-04-26
 */

#ifndef COMM_RECV_H
#define COMM_RECV_H

/**
 * @brief
 *
 * @param threadparam
 * @return void*
 */
void* comm_recv_task_callback(void *threadparam);


#endif/**
 * @brief
 *
 * @file readConfiguration.h
 * @author Gunj Manseta
 * @date 2018-03-17
 */

#ifndef READCONIFG_H
#define READCONIFG_H

/**
 * @brief
 *
 * @return char*
 */
char* configdata_getLogpath();

/**
 * @brief
 *
 * @return uint32_t
 */
uint32_t configdata_getSetupTime();

/**
 * @brief
 *
 * @return uint32_t
 */
uint32_t configdata_getAliveTimeout();

/**
 * @brief Should be called in the main task at the beginning
```

```
 *
 * @return int
 */
int configdata_setup();

/**
 * @brief Should be called at teh end of main task. If not called, memory
leak will occur
 *
 */
void configdata_flush();


#endif/**
 * @brief
 *
 * @file common_helper.h
 * @author Gunj Manseta
 * @date 2018-03-09
 */
#ifndef COMMON_HELPER_H
#define COMMON_HELPER_H

#include <mqueue.h>
#include <pthread.h>

#include "posixTimer.h"


typedef enum
{
    LOGGER_TASK_ID = 0,
    TEMPERATURE_TASK_ID,
    SOCKET_TASK_ID,
    LIGHT_TASK_ID,
    COMM_RECEIVER_ID,
    COMM_SENDER_ID,
    DISPATCHER_TASK_ID,
    MAIN_TASK_ID            //This MAINT TASK should alwys be on the last

}TASK_IDENTIFIER_T;

#define NUM_CHILD_THREADS (MAIN_TASK_ID)

volatile int aliveStatus[NUM_CHILD_THREADS];

pthread_mutex_t aliveState_lock;


pthread_t pthread_id[NUM_CHILD_THREADS];

mqd_t get_queue_handle(TASK_IDENTIFIER_T taskid);

pthread_barrier_t tasks_barrier;


extern const char* const task_identifier_string[NUM_CHILD_THREADS+1];
```

```
/**
 * @brief Get the Task Identfier String
 *
 * @param taskid
 * @return const char*
 */
static inline const char* getTaskIdentfierString(TASK_IDENTIFIER_T
taskid)
{
    return task_identifier_string[taskid];
}


/**
 * @brief Registers a timer, addigns the handler and starts it
 *
 * @param timer_id
 * @param usec
 * @param oneshot
 * @param timer_handler
 * @param handlerArgs
 * @return int
 */
int register_and_start_timer(timer_t *timer_id, uint32_t usec, uint8_t
oneshot, void (*timer_handler)(union sigval), void *handlerArgs);


#endif/**
 * @brief
 *
 * @file error_data.h
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#ifndef ERROR_DATA_H
#define ERROR_DATA_H

#include <unistd.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <stdio.h>


typedef enum{

    ERR     = -1,
    SUCCESS = 0,
}RETURN_T;

//syscall(SYS_gettid) [TID:%ld] ",getpid()

//#define LOG_STDOUT(format, ...) printf("[PID:%d]",getpid());
printf(format, ##__VA_ARGS__)
#define LOG_STDOUT(format, ...)
do{printf("[PID:%d][TID:%ld]",getpid(),syscall(SYS_gettid));
printf(format, ##__VA_ARGS__); fflush(stdout);}while(0)
```

```
#define LOG_STDOUT_COMM(recv_comm_msg)    \
            ({LOG_STDOUT(INFO "\n*******\
            \nSRCID:%u, SRC_BRDID:%u, DST_ID:%u, DST_BRDID:%u MSGID:%u\
            \nSensorVal: %.2f MSG:%s\
            \nChecksum:%u ?= %u\n********\n",\
            recv_comm_msg.src_id, recv_comm_msg.src_brd_id,
recv_comm_msg.dst_id,recv_comm_msg.dst_brd_id,recv_comm_msg.msg_id,recv_c
omm_msg.data.distance_cm,recv_comm_msg.message,recv_comm_msg.checksum,
check );})

#define ERROR   "[ERROR] "
#define INFO    "[INFO] "
#define SIGNAL  "[SIGNAL] "
#define WARNING "[WARNING] "


#endif/*
 * Copyright 2014 Luis Pabon, Jr.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

/*
 * Programming by Contract is a programming methodology
 * which binds the caller and the function called to a
 * contract. The contract is represented using Hoare Triple:
 *      {P} C {Q}
 * where {P} is the precondition before executing command C,
 * and {Q} is the postcondition.
 *
 * See also:
 * http://en.wikipedia.org/wiki/Design_by_contract
 * http://en.wikipedia.org/wiki/Hoare_logic
 * http://dlang.org/dbc.html
 */
#ifndef CMOCKA_PBC_H_
#define CMOCKA_PBC_H_

#if defined(UNIT_TESTING) || defined (DEBUG)

#include <assert.h>

/*
 * Checks caller responsibility against contract
 */
#define REQUIRE(cond) assert(cond)
```

```
/*
 * Checks function reponsability against contract.
 */
#define ENSURE(cond) assert(cond)

/*
 * While REQUIRE and ENSURE apply to functions, INVARIANT
 * applies to classes/structs.  It ensures that intances
 * of the class/struct are consistent. In other words,
 * that the instance has not been corrupted.
 */
#define INVARIANT(invariant_fnc) do{ (invariant_fnc) } while (0);

#else
#define REQUIRE(cond) do { } while (0);
#define ENSURE(cond) do { } while (0);
#define INVARIANT(invariant_fnc) do{ } while (0);

#endif /* defined(UNIT_TESTING) || defined (DEBUG) */
#endif /* CMOCKA_PBC_H_ */

/*
 * Copyright 2008 Google Inc.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 * http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */
#ifndef CMOCKA_H_
#define CMOCKA_H_

#ifdef _WIN32
# ifdef _MSC_VER

#define __func__ __FUNCTION__

# ifndef inline
#define inline __inline
# endif /* inline */

#  if _MSC_VER < 1500
#   ifdef __cplusplus
extern "C" {
#   endif   /* __cplusplus */
int __stdcall IsDebuggerPresent();
#   ifdef __cplusplus
} /* extern "C" */
#   endif   /* __cplusplus */
#  endif  /* _MSC_VER < 1500 */
```

```
# endif /* _MSC_VER */
#endif  /* _WIN32 */

/**
 * @defgroup cmocka The CMocka API
 *
 * These headers or their equivalents should be included prior to
including
 * this header file.
 * @code
 * #include <stdarg.h>
 * #include <stddef.h>
 * #include <setjmp.h>
 * @endcode
 *
 * This allows test applications to use custom definitions of C standard
 * library functions and types.
 *
 * @{
 */

/* If __WORDSIZE is not set, try to figure it out and default to 32 bit.
*/
#ifndef __WORDSIZE
# if defined(__x86_64__) && !defined(__ILP32__)
#  define __WORDSIZE 64
# else
#  define __WORDSIZE 32
# endif
#endif

#ifdef DOXYGEN
/**
 * Largest integral type.  This type should be large enough to hold any
 * pointer or integer supported by the compiler.
 */
typedef uintmax_t LargestIntegralType;
#else /* DOXGEN */
#ifndef LargestIntegralType
# if __WORDSIZE == 64
#  define LargestIntegralType unsigned long int
# else
#  define LargestIntegralType unsigned long long int
# endif
#endif /* LargestIntegralType */
#endif /* DOXYGEN */

/* Printf format used to display LargestIntegralType as a hexidecimal. */
#ifndef LargestIntegralTypePrintfFormat
# ifdef _WIN32
#  define LargestIntegralTypePrintfFormat "0x%I64x"
# else
#  if __WORDSIZE == 64
#   define LargestIntegralTypePrintfFormat "%#lx"
#  else
#   define LargestIntegralTypePrintfFormat "%#llx"
#  endif
# endif /* _WIN32 */
```

```
#endif /* LargestIntegralTypePrintfFormat */

/* Printf format used to display LargestIntegralType as a decimal. */
#ifndef LargestIntegralTypePrintfFormatDecimal
# ifdef _WIN32
#  define LargestIntegralTypePrintfFormatDecimal "%I64u"
# else
#  if __WORDSIZE == 64
#   define LargestIntegralTypePrintfFormatDecimal "%lu"
#  else
#   define LargestIntegralTypePrintfFormatDecimal "%llu"
#  endif
# endif /* _WIN32 */
#endif /* LargestIntegralTypePrintfFormat */

/* Perform an unsigned cast to LargestIntegralType. */
#define cast_to_largest_integral_type(value) \
    ((LargestIntegralType)(value))

/* Smallest integral type capable of holding a pointer. */
#if !defined(_UINTPTR_T) && !defined(_UINTPTR_T_DEFINED)
# if defined(_WIN32)
    /* WIN32 is an ILP32 platform */
    typedef unsigned int uintptr_t;
# elif defined(_WIN64)
    typedef unsigned long int uintptr_t
# else /* _WIN32 */

/* ILP32 and LP64 platforms */
#  ifdef __WORDSIZE /* glibc */
#   if __WORDSIZE == 64
      typedef unsigned long int uintptr_t;
#   else
      typedef unsigned int uintptr_t;
#   endif /* __WORDSIZE == 64 */
#  else /* __WORDSIZE */
#   if defined(_LP64) || defined(_I32LPx)
      typedef unsigned long int uintptr_t;
#   else
      typedef unsigned int uintptr_t;
#   endif
#  endif /* __WORDSIZE */
# endif /* _WIN32 */

# define _UINTPTR_T
# define _UINTPTR_T_DEFINED
#endif /* !defined(_UINTPTR_T) || !defined(_UINTPTR_T_DEFINED) */

/* Perform an unsigned cast to uintptr_t. */
#define cast_to_pointer_integral_type(value) \
    ((uintptr_t)((size_t)(value)))

/* Perform a cast of a pointer to LargestIntegralType */
#define cast_ptr_to_largest_integral_type(value) \
cast_to_largest_integral_type(cast_to_pointer_integral_type(value))

/* GCC have printf type attribute check.  */
#ifdef __GNUC__
```

```
#define CMOCKA_PRINTF_ATTRIBUTE(a,b) \
    __attribute__ ((__format__ (__printf__, a, b)))
#else
#define CMOCKA_PRINTF_ATTRIBUTE(a,b)
#endif /* __GNUC__ */

#if defined(__GNUC__)
#define CMOCKA_DEPRECATED __attribute__ ((deprecated))
#elif defined(_MSC_VER)
#define CMOCKA_DEPRECATED __declspec(deprecated)
#else
#define CMOCKA_DEPRECATED
#endif

#define WILL_RETURN_ALWAYS -1
#define WILL_RETURN_ONCE -2

/**
 * @defgroup cmocka_mock Mock Objects
 * @ingroup cmocka
 *
 * Mock objects mock objects are simulated objects that mimic the
behavior of
 * real objects. Instead of calling the real objects, the tested object
calls a
 * mock object that merely asserts that the correct methods were called,
with
 * the expected parameters, in the correct order.
 *
 * <ul>
 * <li><strong>will_return(function, value)</strong> - The will_return()
macro
 * pushes a value onto a stack of mock values. This macro is intended to
be
 * used by the unit test itself, while programming the behaviour of the
mocked
 * object.</li>
 *
 * <li><strong>mock()</strong> - the mock macro pops a value from a stack
of
 * test values. The user of the mock() macro is the mocked object that
uses it
 * to learn how it should behave.</li>
 * </ul>
 *
 * Because the will_return() and mock() are intended to be used in pairs,
the
 * cmocka library would fail the test if there are more values pushed
onto the
 * stack using will_return() than consumed with mock() and vice-versa.
 *
 * The following unit test stub illustrates how would a unit test
instruct the
 * mock object to return a particular value:
 *
 * @code
 * will_return(chef_cook, "hotdog");
 * will_return(chef_cook, 0);
```

```
 * @endcode
 *
 * Now the mock object can check if the parameter it received is the
parameter
 * which is expected by the test driver. This can be done the following
way:
 *
 * @code
 * int chef_cook(const char *order, char **dish_out)
 * {
 *     check_expected(order);
 * }
 * @endcode
 *
 * For a complete example please at a look
 * <a
href="http://git.cryptomilk.org/projects/cmocka.git/tree/example/chef_wra
p/waiter_test_wrap.c">here</a>.
 *
 * @{
 */

#ifdef DOXYGEN
/**
 * @brief Retrieve a return value of the current function.
 *
 * @return The value which was stored to return by this function.
 *
 * @see will_return()
 */
LargestIntegralType mock(void);
#else
#define mock() _mock(__func__, __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Retrieve a typed return value of the current function.
 *
 * The value would be casted to type internally to avoid having the
 * caller to do the cast manually.
 *
 * @param[in]  #type  The expected type of the return value
 *
 * @return The value which was stored to return by this function.
 *
 * @code
 * int param;
 *
 * param = mock_type(int);
 * @endcode
 *
 * @see will_return()
 * @see mock()
 * @see mock_ptr_type()
 */
#type mock_type(#type);
#else
```

```
#define mock_type(type) ((type) mock())
#endif

#ifdef DOXYGEN
/**
 * @brief Retrieve a typed return value of the current function.
 *
 * The value would be casted to type internally to avoid having the
 * caller to do the cast manually but also casted to uintptr_t to make
 * sure the result has a valid size to be used as a pointer.
 *
 * @param[in]  #type  The expected type of the return value
 *
 * @return The value which was stored to return by this function.
 *
 * @code
 * char *param;
 *
 * param = mock_ptr_type(char *);
 * @endcode
 *
 * @see will_return()
 * @see mock()
 * @see mock_type()
 */
type mock_ptr_type(#type);
#else
#define mock_ptr_type(type) ((type) (uintptr_t) mock())
#endif


#ifdef DOXYGEN
/**
 * @brief Store a value to be returned by mock() later.
 *
 * @param[in]  #function  The function which should return the given
value.
 *
 * @param[in]  value The value to be returned by mock().
 *
 * @code
 * int return_integer(void)
 * {
 *      return (int)mock();
 * }
 *
 * static void test_integer_return(void **state)
 * {
 *      will_return(return_integer, 42);
 *
 *      assert_int_equal(my_function_calling_return_integer(), 42);
 * }
 * @endcode
 *
 * @see mock()
 * @see will_return_count()
 */
void will_return(#function, LargestIntegralType value);
```

```
#else
#define will_return(function, value) \
    _will_return(#function, __FILE__, __LINE__, \
                 cast_to_largest_integral_type(value), 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Store a value to be returned by mock() later.
 *
 * @param[in]  #function  The function which should return the given
value.
 *
 * @param[in]  value The value to be returned by mock().
 *
 * @param[in]  count The parameter indicates the number of times the
value should
 *                   be returned by mock(). If count is set to -1, the
value
 *                   will always be returned but must be returned at
least once.
 *                   If count is set to -2, the value will always be
returned
 *                   by mock(), but is not required to be returned.
 *
 * @see mock()
 */
void will_return_count(#function, LargestIntegralType value, int count);
#else
#define will_return_count(function, value, count) \
    _will_return(#function, __FILE__, __LINE__, \
                 cast_to_largest_integral_type(value), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Store a value that will be always returned by mock().
 *
 * @param[in]  #function  The function which should return the given
value.
 *
 * @param[in]  #value The value to be returned by mock().
 *
 * This is equivalent to:
 * @code
 * will_return_count(function, value, -1);
 * @endcode
 *
 * @see will_return_count()
 * @see mock()
 */
void will_return_always(#function, LargestIntegralType value);
#else
#define will_return_always(function, value) \
    will_return_count(function, (value), WILL_RETURN_ALWAYS)
#endif

#ifdef DOXYGEN
```

```
/**
 * @brief Store a value that may be always returned by mock().
 *
 * This stores a value which will always be returned by mock() but is not
 * required to be returned by at least one call to mock(). Therefore,
 * in contrast to will_return_always() which causes a test failure if it
 * is not returned at least once, will_return_maybe() will never cause a
test
 * to fail if its value is not returned.
 *
 * @param[in]  #function  The function which should return the given
value.
 *
 * @param[in]  #value The value to be returned by mock().
 *
 * This is equivalent to:
 * @code
 * will_return_count(function, value, -2);
 * @endcode
 *
 * @see will_return_count()
 * @see mock()
 */
void will_return_maybe(#function, LargestIntegralType value);
#else
#define will_return_maybe(function, value) \
    will_return_count(function, (value), WILL_RETURN_ONCE)
#endif
/** @} */

/**
 * @defgroup cmocka_param Checking Parameters
 * @ingroup cmocka
 *
 * Functionality to store expected values for mock function parameters.
 *
 * In addition to storing the return values of mock functions, cmocka
provides
 * functionality to store expected values for mock function parameters
using
 * the expect_*() functions provided. A mock function parameter can then
be
 * validated using the check_expected() macro.
 *
 * Successive calls to expect_*() macros for a parameter queues values to
check
 * the specified parameter. check_expected() checks a function parameter
 * against the next value queued using expect_*(), if the parameter check
fails
 * a test failure is signalled. In addition if check_expected() is called
and
 * no more parameter values are queued a test failure occurs.
 *
 * The following test stub illustrates how to do this. First is the the
function
 * we call in the test driver:
 *
 * @code
```

```
 * static void test_driver(void **state)
 * {
 *     expect_string(chef_cook, order, "hotdog");
 * }
 * @endcode
 *
 * Now the chef_cook function can check if the parameter we got passed is
the
 * parameter which is expected by the test driver. This can be done the
 * following way:
 *
 * @code
 * int chef_cook(const char *order, char **dish_out)
 * {
 *     check_expected(order);
 * }
 * @endcode
 *
 * For a complete example please at a look at
 * <a
href="http://git.cryptomilk.org/projects/cmocka.git/tree/example/chef_wra
p/waiter_test_wrap.c">here</a>
 *
 * @{
 */

/*
 * Add a custom parameter checking function.  If the event parameter is
NULL
 * the event structure is allocated internally by this function.  If
event
 * parameter is provided it must be allocated on the heap and doesn't
need to
 * be deallocated by the caller.
 */
#ifdef DOXYGEN
/**
 * @brief Add a custom parameter checking function.
 *
 * If the event parameter is NULL the event structure is allocated
internally
 * by this function. If the parameter is provided it must be allocated on
the
 * heap and doesn't need to be deallocated by the caller.
 *
 * @param[in]  #function  The function to add a custom parameter checking
 *                        function for.
 *
 * @param[in]  #parameter The parameters passed to the function.
 *
 * @param[in]  #check_function  The check function to call.
 *
 * @param[in]  check_data       The data to pass to the check function.
 */
void expect_check(#function, #parameter, #check_function, const void
*check_data);
#else
#define expect_check(function, parameter, check_function, check_data) \
```

```
    _expect_check(#function, #parameter, __FILE__, __LINE__,
check_function, \
                  cast_to_largest_integral_type(check_data), NULL, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is part of the
provided
 *        array.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value_array[] The array to check for the value.
 *
 * @see check_expected().
 */
void expect_in_set(#function, #parameter, LargestIntegralType
value_array[]);
#else
#define expect_in_set(function, parameter, value_array) \
    expect_in_set_count(function, parameter, value_array, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is part of the
provided
 *        array.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value_array[] The array to check for the value.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_in_set_count(#function, #parameter, LargestIntegralType
value_array[], size_t count);
#else
#define expect_in_set_count(function, parameter, value_array, count) \
```

```
    _expect_in_set(#function, #parameter, __FILE__, __LINE__,
value_array, \
                   sizeof(value_array) / sizeof((value_array)[0]), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is not part of the
 *        provided array.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value_array[] The array to check for the value.
 *
 * @see check_expected().
 */
void expect_not_in_set(#function, #parameter, LargestIntegralType
value_array[]);
#else
#define expect_not_in_set(function, parameter, value_array) \
    expect_not_in_set_count(function, parameter, value_array, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is not part of the
 *        provided array.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value_array[] The array to check for the value.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_not_in_set_count(#function, #parameter, LargestIntegralType
value_array[], size_t count);
#else
#define expect_not_in_set_count(function, parameter, value_array, count)
\
    _expect_not_in_set( \
```

```
        #function, #parameter, __FILE__, __LINE__, value_array, \
        sizeof(value_array) / sizeof((value_array)[0]), count)
#endif


#ifdef DOXYGEN
/**
 * @brief Add an event to check a parameter is inside a numerical range.
 * The check would succeed if minimum <= value <= maximum.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  minimum  The lower boundary of the interval to check
against.
 *
 * @param[in]  maximum  The upper boundary of the interval to check
against.
 *
 * @see check_expected().
 */
void expect_in_range(#function, #parameter, LargestIntegralType minimum,
LargestIntegralType maximum);
#else
#define expect_in_range(function, parameter, minimum, maximum) \
    expect_in_range_count(function, parameter, minimum, maximum, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check a parameter is inside a
 * numerical range. The check would succeed if minimum <= value <=
maximum.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  minimum  The lower boundary of the interval to check
against.
 *
 * @param[in]  maximum  The upper boundary of the interval to check
against.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
```

```
 *
 * @see check_expected().
 */
void expect_in_range_count(#function, #parameter, LargestIntegralType
minimum, LargestIntegralType maximum, size_t count);
#else
#define expect_in_range_count(function, parameter, minimum, maximum,
count) \
    _expect_in_range(#function, #parameter, __FILE__, __LINE__, minimum,
\
                     maximum, count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check a parameter is outside a numerical range.
 * The check would succeed if minimum > value > maximum.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  minimum  The lower boundary of the interval to check
against.
 *
 * @param[in]  maximum  The upper boundary of the interval to check
against.
 *
 * @see check_expected().
 */
void expect_not_in_range(#function, #parameter, LargestIntegralType
minimum, LargestIntegralType maximum);
#else
#define expect_not_in_range(function, parameter, minimum, maximum) \
    expect_not_in_range_count(function, parameter, minimum, maximum, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check a parameter is outside a
 * numerical range. The check would succeed if minimum > value > maximum.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  minimum  The lower boundary of the interval to check
against.
 *
```

```
 * @param[in]  maximum  The upper boundary of the interval to check
against.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_not_in_range_count(#function, #parameter, LargestIntegralType
minimum, LargestIntegralType maximum, size_t count);
#else
#define expect_not_in_range_count(function, parameter, minimum, maximum,
\
                                  count) \
    _expect_not_in_range(#function, #parameter, __FILE__, __LINE__, \
                         minimum, maximum, count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if a parameter is the given value.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value  The value to check.
 *
 * @see check_expected().
 */
void expect_value(#function, #parameter, LargestIntegralType value);
#else
#define expect_value(function, parameter, value) \
    expect_value_count(function, parameter, value, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check if a parameter is the given
value.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value  The value to check.
 *
```

```
 * @param[in]  count  The count parameter returns the number of times the
value
 *                   should be returned by check_expected(). If count is
set
 *                   to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_value_count(#function, #parameter, LargestIntegralType value,
size_t count);
#else
#define expect_value_count(function, parameter, value, count) \
    _expect_value(#function, #parameter, __FILE__, __LINE__, \
                  cast_to_largest_integral_type(value), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if a parameter isn't the given value.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value  The value to check.
 *
 * @see check_expected().
 */
void expect_not_value(#function, #parameter, LargestIntegralType value);
#else
#define expect_not_value(function, parameter, value) \
    expect_not_value_count(function, parameter, value, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check if a parameter isn't the given
value.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  value  The value to check.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                   should be returned by check_expected(). If count is
set
 *                   to -1 the value will always be returned.
```

```
 *
 * @see check_expected().
 */
void expect_not_value_count(#function, #parameter, LargestIntegralType
value, size_t count);
#else
#define expect_not_value_count(function, parameter, value, count) \
    _expect_not_value(#function, #parameter, __FILE__, __LINE__, \
                      cast_to_largest_integral_type(value), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is equal to the
 *        provided string.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  string   The string value to compare.
 *
 * @see check_expected().
 */
void expect_string(#function, #parameter, const char *string);
#else
#define expect_string(function, parameter, string) \
    expect_string_count(function, parameter, string, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value is equal to the
 *        provided string.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  string   The string value to compare.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
```

```
void expect_string_count(#function, #parameter, const char *string,
size_t count);
#else
#define expect_string_count(function, parameter, string, count) \
    _expect_string(#function, #parameter, __FILE__, __LINE__, \
                   (const char*)(string), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value isn't equal to the
 *        provided string.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  string   The string value to compare.
 *
 * @see check_expected().
 */
void expect_not_string(#function, #parameter, const char *string);
#else
#define expect_not_string(function, parameter, string) \
    expect_not_string_count(function, parameter, string, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter value isn't equal to the
 *        provided string.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  string   The string value to compare.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_not_string_count(#function, #parameter, const char *string,
size_t count);
#else
#define expect_not_string_count(function, parameter, string, count) \
```

```
    _expect_not_string(#function, #parameter, __FILE__, __LINE__, \
                       (const char*)(string), count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter does match an area of
memory.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  memory  The memory to compare.
 *
 * @param[in]  size  The size of the memory to compare.
 *
 * @see check_expected().
 */
void expect_memory(#function, #parameter, void *memory, size_t size);
#else
#define expect_memory(function, parameter, memory, size) \
    expect_memory_count(function, parameter, memory, size, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check if the parameter does match an
area
 *        of memory.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  memory  The memory to compare.
 *
 * @param[in]  size  The size of the memory to compare.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_memory_count(#function, #parameter, void *memory, size_t
size, size_t count);
#else
```

```
#define expect_memory_count(function, parameter, memory, size, count) \
    _expect_memory(#function, #parameter, __FILE__, __LINE__, \
                   (const void*)(memory), size, count)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to check if the parameter doesn't match an area of
 *        memory.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  memory  The memory to compare.
 *
 * @param[in]  size  The size of the memory to compare.
 *
 * @see check_expected().
 */
void expect_not_memory(#function, #parameter, void *memory, size_t size);
#else
#define expect_not_memory(function, parameter, memory, size) \
    expect_not_memory_count(function, parameter, memory, size, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check if the parameter doesn't match
an
 *        area of memory.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  memory  The memory to compare.
 *
 * @param[in]  size  The size of the memory to compare.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_not_memory_count(#function, #parameter, void *memory, size_t
size, size_t count);
```

```
#else
#define expect_not_memory_count(function, parameter, memory, size, count)
\
    _expect_not_memory(#function, #parameter, __FILE__, __LINE__, \
                       (const void*)(memory), size, count)
#endif


#ifdef DOXYGEN
/**
 * @brief Add an event to check if a parameter (of any value) has been
passed.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @see check_expected().
 */
void expect_any(#function, #parameter);
#else
#define expect_any(function, parameter) \
    expect_any_count(function, parameter, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Add an event to repeatedly check if a parameter (of any value)
has
 *        been passed.
 *
 * The event is triggered by calling check_expected() in the mocked
function.
 *
 * @param[in]  #function  The function to add the check for.
 *
 * @param[in]  #parameter The name of the parameter passed to the
function.
 *
 * @param[in]  count  The count parameter returns the number of times the
value
 *                    should be returned by check_expected(). If count is
set
 *                    to -1 the value will always be returned.
 *
 * @see check_expected().
 */
void expect_any_count(#function, #parameter, size_t count);
#else
#define expect_any_count(function, parameter, count) \
    _expect_any(#function, #parameter, __FILE__, __LINE__, count)
#endif

#ifdef DOXYGEN
```

```
/**
 * @brief Determine whether a function parameter is correct.
 *
 * This ensures the next value queued by one of the expect_*() macros
matches
 * the specified variable.
 *
 * This function needs to be called in the mock object.
 *
 * @param[in]  #parameter  The parameter to check.
 */
void check_expected(#parameter);
#else
#define check_expected(parameter) \
    _check_expected(__func__, #parameter, __FILE__, __LINE__, \
                    cast_to_largest_integral_type(parameter))
#endif

#ifdef DOXYGEN
/**
 * @brief Determine whether a function parameter is correct.
 *
 * This ensures the next value queued by one of the expect_*() macros
matches
 * the specified variable.
 *
 * This function needs to be called in the mock object.
 *
 * @param[in]  #parameter  The pointer to check.
 */
void check_expected_ptr(#parameter);
#else
#define check_expected_ptr(parameter) \
    _check_expected(__func__, #parameter, __FILE__, __LINE__, \
                    cast_ptr_to_largest_integral_type(parameter))
#endif

/** @} */

/**
 * @defgroup cmocka_asserts Assert Macros
 * @ingroup cmocka
 *
 * This is a set of useful assert macros like the standard C libary's
 * assert(3) macro.
 *
 * On an assertion failure a cmocka assert macro will write the failure
to the
 * standard error stream and signal a test failure. Due to limitations of
the C
 * language the general C standard library assert() and cmocka's
assert_true()
 * and assert_false() macros can only display the expression that caused
the
 * assert failure. cmocka's type specific assert macros,
assert_{type}_equal()
 * and assert_{type}_not_equal(), display the data that caused the
assertion
```

```
 * failure which increases data visibility aiding debugging of failing
test
 * cases.
 *
 * @{
 */

#ifdef DOXYGEN
/**
 * @brief Assert that the given expression is true.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if expression is false (i.e., compares equal to
 * zero).
 *
 * @param[in]  expression  The expression to evaluate.
 *
 * @see assert_int_equal()
 * @see assert_string_equal()
 */
void assert_true(scalar expression);
#else
#define assert_true(c) _assert_true(cast_to_largest_integral_type(c), #c,
\
                                    __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the given expression is false.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if expression is true.
 *
 * @param[in]  expression  The expression to evaluate.
 *
 * @see assert_int_equal()
 * @see assert_string_equal()
 */
void assert_false(scalar expression);
#else
#define assert_false(c) _assert_true(!(cast_to_largest_integral_type(c)),
#c, \
                                     __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the return_code is greater than or equal to 0.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the return code is smaller than 0. If the
function
 * you check sets an errno if it fails you can pass it to the function
and
```

```
 * it will be printed as part of the error message.
 *
 * @param[in]  rc       The return code to evaluate.
 *
 * @param[in]  error    Pass errno here or 0.
 */
void assert_return_code(int rc, int error);
#else
#define assert_return_code(rc, error) \
    _assert_return_code(cast_to_largest_integral_type(rc), \
                        sizeof(rc), \
                        cast_to_largest_integral_type(error), \
                        #rc, __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the given pointer is non-NULL.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the pointer is non-NULL.
 *
 * @param[in]  pointer  The pointer to evaluate.
 *
 * @see assert_null()
 */
void assert_non_null(void *pointer);
#else
#define assert_non_null(c)
_assert_true(cast_ptr_to_largest_integral_type(c), #c, \
                                          __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the given pointer is NULL.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the pointer is non-NULL.
 *
 * @param[in]  pointer  The pointer to evaluate.
 *
 * @see assert_non_null()
 */
void assert_null(void *pointer);
#else
#define assert_null(c)
_assert_true(!(cast_ptr_to_largest_integral_type(c)), #c, \
__FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given pointers are equal.
 *
```

```
 * The function prints an error message and terminates the test by
calling
 * fail() if the pointers are not equal.
 *
 * @param[in]  a        The first pointer to compare.
 *
 * @param[in]  b        The pointer to compare against the first one.
 */
void assert_ptr_equal(void *a, void *b);
#else
#define assert_ptr_equal(a, b) \
    _assert_int_equal(cast_ptr_to_largest_integral_type(a), \
                      cast_ptr_to_largest_integral_type(b), \
                      __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given pointers are not equal.
 *
 * The function prints an error message and terminates the test by
calling
 * fail() if the pointers are equal.
 *
 * @param[in]  a        The first pointer to compare.
 *
 * @param[in]  b        The pointer to compare against the first one.
 */
void assert_ptr_not_equal(void *a, void *b);
#else
#define assert_ptr_not_equal(a, b) \
    _assert_int_not_equal(cast_ptr_to_largest_integral_type(a), \
                          cast_ptr_to_largest_integral_type(b), \
                          __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given integers are equal.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the integers are not equal.
 *
 * @param[in]  a  The first integer to compare.
 *
 * @param[in]  b  The integer to compare against the first one.
 */
void assert_int_equal(int a, int b);
#else
#define assert_int_equal(a, b) \
    _assert_int_equal(cast_to_largest_integral_type(a), \
                      cast_to_largest_integral_type(b), \
                      __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
```

```
 * @brief Assert that the two given integers are not equal.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the integers are equal.
 *
 * @param[in]  a  The first integer to compare.
 *
 * @param[in]  b  The integer to compare against the first one.
 *
 * @see assert_int_equal()
 */
void assert_int_not_equal(int a, int b);
#else
#define assert_int_not_equal(a, b) \
    _assert_int_not_equal(cast_to_largest_integral_type(a), \
                          cast_to_largest_integral_type(b), \
                          __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given strings are equal.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the strings are not equal.
 *
 * @param[in]  a  The string to check.
 *
 * @param[in]  b  The other string to compare.
 */
void assert_string_equal(const char *a, const char *b);
#else
#define assert_string_equal(a, b) \
    _assert_string_equal((const char*)(a), (const char*)(b), __FILE__, \
                         __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given strings are not equal.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the strings are equal.
 *
 * @param[in]  a  The string to check.
 *
 * @param[in]  b  The other string to compare.
 */
void assert_string_not_equal(const char *a, const char *b);
#else
#define assert_string_not_equal(a, b) \
    _assert_string_not_equal((const char*)(a), (const char*)(b),
__FILE__, \
                             __LINE__)
#endif
```

```
#ifdef DOXYGEN
/**
 * @brief Assert that the two given areas of memory are equal, otherwise
fail.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the memory is not equal.
 *
 * @param[in]  a  The first memory area to compare
 *                (interpreted as unsigned char).
 *
 * @param[in]  b  The second memory area to compare
 *                (interpreted as unsigned char).
 *
 * @param[in]  size  The first n bytes of the memory areas to compare.
 */
void assert_memory_equal(const void *a, const void *b, size_t size);
#else
#define assert_memory_equal(a, b, size) \
    _assert_memory_equal((const void*)(a), (const void*)(b), size,
__FILE__, \
                         __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the two given areas of memory are not equal.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if the memory is equal.
 *
 * @param[in]  a  The first memory area to compare
 *                (interpreted as unsigned char).
 *
 * @param[in]  b  The second memory area to compare
 *                (interpreted as unsigned char).
 *
 * @param[in]  size  The first n bytes of the memory areas to compare.
 */
void assert_memory_not_equal(const void *a, const void *b, size_t size);
#else
#define assert_memory_not_equal(a, b, size) \
    _assert_memory_not_equal((const void*)(a), (const void*)(b), size, \
                             __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the specified value is not smaller than the minimum
 * and and not greater than the maximum.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if value is not in range.
 *
```

```
 * @param[in]  value  The value to check.
 *
 * @param[in]  minimum  The minimum value allowed.
 *
 * @param[in]  maximum  The maximum value allowed.
 */
void assert_in_range(LargestIntegralType value, LargestIntegralType
minimum, LargestIntegralType maximum);
#else
#define assert_in_range(value, minimum, maximum) \
    _assert_in_range( \
        cast_to_largest_integral_type(value), \
        cast_to_largest_integral_type(minimum), \
        cast_to_largest_integral_type(maximum), __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the specified value is smaller than the minimum or
 * greater than the maximum.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if value is in range.
 *
 * @param[in]  value  The value to check.
 *
 * @param[in]  minimum  The minimum value to compare.
 *
 * @param[in]  maximum  The maximum value to compare.
 */
void assert_not_in_range(LargestIntegralType value, LargestIntegralType
minimum, LargestIntegralType maximum);
#else
#define assert_not_in_range(value, minimum, maximum) \
    _assert_not_in_range( \
        cast_to_largest_integral_type(value), \
        cast_to_largest_integral_type(minimum), \
        cast_to_largest_integral_type(maximum), __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the specified value is within a set.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if value is not within a set.
 *
 * @param[in]  value  The value to look up
 *
 * @param[in]  values[]  The array to check for the value.
 *
 * @param[in]  count  The size of the values array.
 */
void assert_in_set(LargestIntegralType value, LargestIntegralType
values[], size_t count);
#else
```

```
#define assert_in_set(value, values, number_of_values) \
    _assert_in_set(value, values, number_of_values, __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Assert that the specified value is not within a set.
 *
 * The function prints an error message to standard error and terminates
the
 * test by calling fail() if value is within a set.
 *
 * @param[in]  value  The value to look up
 *
 * @param[in]  values[]  The array to check for the value.
 *
 * @param[in]  count  The size of the values array.
 */
void assert_not_in_set(LargestIntegralType value, LargestIntegralType
values[], size_t count);
#else
#define assert_not_in_set(value, values, number_of_values) \
    _assert_not_in_set(value, values, number_of_values, __FILE__,
__LINE__)
#endif

/** @} */

/**
 * @defgroup cmocka_call_order Call Ordering
 * @ingroup cmocka
 *
 * It is often beneficial to  make sure that functions are called in an
 * order. This is independent of mock returns and parameter checking as
both
 * of the aforementioned do not check the order in which they are called
from
 * different functions.
 *
 * <ul>
 * <li><strong>expect_function_call(function)</strong> - The
 * expect_function_call() macro pushes an expectation onto the stack of
 * expected calls.</li>
 *
 * <li><strong>function_called()</strong> - pops a value from the stack
of
 * expected calls. function_called() is invoked within the mock object
 * that uses it.
 * </ul>
 *
 * expect_function_call() and function_called() are intended to be used
in
 * pairs. Cmocka will fail a test if there are more or less expected
calls
 * created (e.g. expect_function_call()) than consumed with
function_called().
 * There are provisions such as ignore_function_calls() which allow this
```

```
 * restriction to be circumvented in tests where mock calls for the code
under
 * test are not the focus of the test.
 *
 * The following example illustrates how a unit test instructs cmocka
 * to expect a function_called() from a particular mock,
 * <strong>chef_sing()</strong>:
 *
 * @code
 * void chef_sing(void);
 *
 * void code_under_test()
 * {
 *   chef_sing();
 * }
 *
 * void some_test(void **state)
 * {
 *     expect_function_call(chef_sing);
 *     code_under_test();
 * }
 * @endcode
 *
 * The implementation of the mock then must check whether it was meant to
 * be called by invoking <strong>function_called()</strong>:
 *
 * @code
 * void chef_sing()
 * {
 *     function_called();
 * }
 * @endcode
 *
 * @{
 */

#ifdef DOXYGEN
/**
 * @brief Check that current mocked function is being called in the
expected
 *        order
 *
 * @see expect_function_call()
 */
void function_called(void);
#else
#define function_called() _function_called(__func__, __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Store expected call(s) to a mock to be checked by
function_called()
 *        later.
 *
 * @param[in]  #function  The function which should should be called
 *
 * @param[in]  times number of times this mock must be called
```

```
 *
 * @see function_called()
 */
void expect_function_calls(#function, const int times);
#else
#define expect_function_calls(function, times) \
    _expect_function_call(#function, __FILE__, __LINE__, times)
#endif

#ifdef DOXYGEN
/**
 * @brief Store expected single call to a mock to be checked by
 *        function_called() later.
 *
 * @param[in]  #function  The function which should should be called
 *
 * @see function_called()
 */
void expect_function_call(#function);
#else
#define expect_function_call(function) \
    _expect_function_call(#function, __FILE__, __LINE__, 1)
#endif

#ifdef DOXYGEN
/**
 * @brief Expects function_called() from given mock at least once
 *
 * @param[in]  #function  The function which should should be called
 *
 * @see function_called()
 */
void expect_function_call_any(#function);
#else
#define expect_function_call_any(function) \
    _expect_function_call(#function, __FILE__, __LINE__, -1)
#endif

#ifdef DOXYGEN
/**
 * @brief Ignores function_called() invocations from given mock function.
 *
 * @param[in]  #function  The function which should should be called
 *
 * @see function_called()
 */
void ignore_function_calls(#function);
#else
#define ignore_function_calls(function) \
    _expect_function_call(#function, __FILE__, __LINE__, -2)
#endif

/** @} */

/**
 * @defgroup cmocka_exec Running Tests
 * @ingroup cmocka
 *
```

```
 * This is the way tests are executed with CMocka.
 *
 * The following example illustrates this macro's use with the unit_test
macro.
 *
 * @code
 * void Test0(void **state);
 * void Test1(void **state);
 *
 * int main(void)
 * {
 *     const struct CMUnitTest tests[] = {
 *         cmocka_unit_test(Test0),
 *         cmocka_unit_test(Test1),
 *     };
 *
 *     return cmocka_run_group_tests(tests, NULL, NULL);
 * }
 * @endcode
 *
 * @{
 */

#ifdef DOXYGEN
/**
 * @brief Forces the test to fail immediately and quit.
 */
void fail(void);
#else
#define fail() _fail(__FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Forces the test to not be executed, but marked as skipped
 */
void skip(void);
#else
#define skip() _skip(__FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Forces the test to fail immediately and quit, printing the
reason.
 *
 * @code
 * fail_msg("This is some error message for test");
 * @endcode
 *
 * or
 *
 * @code
 * char *error_msg = "This is some error message for test";
 * fail_msg("%s", error_msg);
 * @endcode
 */
void fail_msg(const char *msg, ...);
```

```
#else
#define fail_msg(msg, ...) do { \
    print_error("ERROR: " msg "\n", ##__VA_ARGS__); \
    fail(); \
} while (0)
#endif

#ifdef DOXYGEN
/**
 * @brief Generic method to run a single test.
 *
 * @deprecated This function was deprecated in favor of
cmocka_run_group_tests
 *
 * @param[in]  #function The function to test.
 *
 * @return 0 on success, 1 if an error occured.
 *
 * @code
 * // A test case that does nothing and succeeds.
 * void null_test_success(void **state) {
 * }
 *
 * int main(void) {
 *      return run_test(null_test_success);
 * }
 * @endcode
 */
int run_test(#function);
#else
#define run_test(f) _run_test(#f, f, NULL, UNIT_TEST_FUNCTION_TYPE_TEST,
NULL)
#endif

static inline void _unit_test_dummy(void **state) {
    (void)state;
}

/** Initializes a UnitTest structure.
 *
 * @deprecated This function was deprecated in favor of cmocka_unit_test
 */
#define unit_test(f) { #f, f, UNIT_TEST_FUNCTION_TYPE_TEST }

#define _unit_test_setup(test, setup) \
    { #test "_" #setup, setup, UNIT_TEST_FUNCTION_TYPE_SETUP }

/** Initializes a UnitTest structure with a setup function.
 *
 * @deprecated This function was deprecated in favor of
cmocka_unit_test_setup
 */
#define unit_test_setup(test, setup) \
    _unit_test_setup(test, setup), \
    unit_test(test), \
    _unit_test_teardown(test, _unit_test_dummy)

#define _unit_test_teardown(test, teardown) \
```

```
    { #test "_" #teardown, teardown, UNIT_TEST_FUNCTION_TYPE_TEARDOWN }

/** Initializes a UnitTest structure with a teardown function.
 *
 * @deprecated This function was deprecated in favor of
cmocka_unit_test_teardown
 */
#define unit_test_teardown(test, teardown) \
    _unit_test_setup(test, _unit_test_dummy), \
    unit_test(test), \
    _unit_test_teardown(test, teardown)

/** Initializes a UnitTest structure for a group setup function.
 *
 * @deprecated This function was deprecated in favor of
cmocka_run_group_tests
 */
#define group_test_setup(setup) \
    { "group_" #setup, setup, UNIT_TEST_FUNCTION_TYPE_GROUP_SETUP }

/** Initializes a UnitTest structure for a group teardown function.
 *
 * @deprecated This function was deprecated in favor of
cmocka_run_group_tests
 */
#define group_test_teardown(teardown) \
    { "group_" #teardown, teardown,
UNIT_TEST_FUNCTION_TYPE_GROUP_TEARDOWN }

/**
 * Initialize an array of UnitTest structures with a setup function for a
test
 * and a teardown function.  Either setup or teardown can be NULL.
 *
 * @deprecated This function was deprecated in favor of
 * cmocka_unit_test_setup_teardown
 */
#define unit_test_setup_teardown(test, setup, teardown) \
    _unit_test_setup(test, setup), \
    unit_test(test), \
    _unit_test_teardown(test, teardown)


/** Initializes a CMUnitTest structure. */
#define cmocka_unit_test(f) { #f, f, NULL, NULL, NULL }

/** Initializes a CMUnitTest structure with a setup function. */
#define cmocka_unit_test_setup(f, setup) { #f, f, setup, NULL, NULL }

/** Initializes a CMUnitTest structure with a teardown function. */
#define cmocka_unit_test_teardown(f, teardown) { #f, f, NULL, teardown,
NULL }

/**
 * Initialize an array of CMUnitTest structures with a setup function for
a test
 * and a teardown function. Either setup or teardown can be NULL.
 */
```

```
#define cmocka_unit_test_setup_teardown(f, setup, teardown) { #f, f,
setup, teardown, NULL }

/**
 * Initialize a CMUnitTest structure with given initial state. It will be
passed
 * to test function as an argument later. It can be used when test state
does
 * not need special initialization or was initialized already.
 * @note If the group setup function initialized the state already, it
won't be
 * overridden by the initial state defined here.
 */
#define cmocka_unit_test_prestate(f, state) { #f, f, NULL, NULL, state }

/**
 * Initialize a CMUnitTest structure with given initial state, setup and
 * teardown function. Any of these values can be NULL. Initial state is
passed
 * later to setup function, or directly to test if none was given.
 * @note If the group setup function initialized the state already, it
won't be
 * overridden by the initial state defined here.
 */
#define cmocka_unit_test_prestate_setup_teardown(f, setup, teardown,
state) { #f, f, setup, teardown, state }

#define run_tests(tests) _run_tests(tests, sizeof(tests) /
sizeof(tests)[0])
#define run_group_tests(tests) _run_group_tests(tests, sizeof(tests) /
sizeof(tests)[0])

#ifdef DOXYGEN
/**
 * @brief Run tests specified by an array of CMUnitTest structures.
 *
 * @param[in]  group_tests[]  The array of unit tests to execute.
 *
 * @param[in]  group_setup    The setup function which should be called
before
 *                            all unit tests are executed.
 *
 * @param[in]  group_teardown The teardown function to be called after
all
 *                            tests have finished.
 *
 * @return 0 on success, or the number of failed tests.
 *
 * @code
 * static int setup(void **state) {
 *      int *answer = malloc(sizeof(int));
 *      if (*answer == NULL) {
 *          return -1;
 *      }
 *      *answer = 42;
 *
 *      *state = answer;
 *
```

```
 *      return 0;
 * }
 *
 * static int teardown(void **state) {
 *      free(*state);
 *
 *      return 0;
 * }
 *
 * static void null_test_success(void **state) {
 *     (void) state;
 * }
 *
 * static void int_test_success(void **state) {
 *      int *answer = *state;
 *      assert_int_equal(*answer, 42);
 * }
 *
 * int main(void) {
 *     const struct CMUnitTest tests[] = {
 *         cmocka_unit_test(null_test_success),
 *         cmocka_unit_test_setup_teardown(int_test_success, setup,
teardown),
 *     };
 *
 *     return cmocka_run_group_tests(tests, NULL, NULL);
 * }
 * @endcode
 *
 * @see cmocka_unit_test
 * @see cmocka_unit_test_setup
 * @see cmocka_unit_test_teardown
 * @see cmocka_unit_test_setup_teardown
 */
int cmocka_run_group_tests(const struct CMUnitTest group_tests[],
                           CMFixtureFunction group_setup,
                           CMFixtureFunction group_teardown);
#else
# define cmocka_run_group_tests(group_tests, group_setup, group_teardown)
\
        _cmocka_run_group_tests(#group_tests, group_tests,
sizeof(group_tests) / sizeof(group_tests)[0], group_setup,
group_teardown)
#endif

#ifdef DOXYGEN
/**
 * @brief Run tests specified by an array of CMUnitTest structures and
specify
 *        a name.
 *
 * @param[in]  group_name     The name of the group test.
 *
 * @param[in]  group_tests[]  The array of unit tests to execute.
 *
 * @param[in]  group_setup    The setup function which should be called
before
 *                            all unit tests are executed.
```

```
 *
 * @param[in]  group_teardown The teardown function to be called after
all
 *                            tests have finished.
 *
 * @return 0 on success, or the number of failed tests.
 *
 * @code
 * static int setup(void **state) {
 *      int *answer = malloc(sizeof(int));
 *      if (*answer == NULL) {
 *          return -1;
 *      }
 *      *answer = 42;
 *
 *      *state = answer;
 *
 *      return 0;
 * }
 *
 * static int teardown(void **state) {
 *      free(*state);
 *
 *      return 0;
 * }
 *
 * static void null_test_success(void **state) {
 *     (void) state;
 * }
 *
 * static void int_test_success(void **state) {
 *      int *answer = *state;
 *      assert_int_equal(*answer, 42);
 * }
 *
 * int main(void) {
 *     const struct CMUnitTest tests[] = {
 *         cmocka_unit_test(null_test_success),
 *         cmocka_unit_test_setup_teardown(int_test_success, setup,
teardown),
 *     };
 *
 *     return cmocka_run_group_tests_name("success_test", tests, NULL,
NULL);
 * }
 * @endcode
 *
 * @see cmocka_unit_test
 * @see cmocka_unit_test_setup
 * @see cmocka_unit_test_teardown
 * @see cmocka_unit_test_setup_teardown
 */
int cmocka_run_group_tests_name(const char *group_name,
                                const struct CMUnitTest group_tests[],
                                CMFixtureFunction group_setup,
                                CMFixtureFunction group_teardown);
#else
```

```
# define cmocka_run_group_tests_name(group_name, group_tests,
group_setup, group_teardown) \
        _cmocka_run_group_tests(group_name, group_tests,
sizeof(group_tests) / sizeof(group_tests)[0], group_setup,
group_teardown)
#endif

/** @} */

/**
 * @defgroup cmocka_alloc Dynamic Memory Allocation
 * @ingroup cmocka
 *
 * Memory leaks, buffer overflows and underflows can be checked using
cmocka.
 *
 * To test for memory leaks, buffer overflows and underflows a module
being
 * tested by cmocka should replace calls to malloc(), calloc() and free()
to
 * test_malloc(), test_calloc() and test_free() respectively. Each time a
block
 * is deallocated using test_free() it is checked for corruption, if a
corrupt
 * block is found a test failure is signalled. All blocks allocated using
the
 * test_*() allocation functions are tracked by the cmocka library. When
a test
 * completes if any allocated blocks (memory leaks) remain they are
reported
 * and a test failure is signalled.
 *
 * For simplicity cmocka currently executes all tests in one process.
Therefore
 * all test cases in a test application share a single address space
which
 * means memory corruption from a single test case could potentially
cause the
 * test application to exit prematurely.
 *
 * @{
 */

#ifdef DOXYGEN
/**
 * @brief Test function overriding malloc.
 *
 * @param[in]  size  The bytes which should be allocated.
 *
 * @return A pointer to the allocated memory or NULL on error.
 *
 * @code
 * #ifdef UNIT_TESTING
 * extern void* _test_malloc(const size_t size, const char* file, const
int line);
 *
 * #define malloc(size) _test_malloc(size, __FILE__, __LINE__)
 * #endif
```

```
 *
 * void leak_memory() {
 *     int * const temporary = (int*)malloc(sizeof(int));
 *     *temporary = 0;
 * }
 * @endcode
 *
 * @see malloc(3)
 */
void *test_malloc(size_t size);
#else
#define test_malloc(size) _test_malloc(size, __FILE__, __LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Test function overriding calloc.
 *
 * The memory is set to zero.
 *
 * @param[in]  nmemb  The number of elements for an array to be
allocated.
 *
 * @param[in]  size   The size in bytes of each array element to
allocate.
 *
 * @return A pointer to the allocated memory, NULL on error.
 *
 * @see calloc(3)
 */
void *test_calloc(size_t nmemb, size_t size);
#else
#define test_calloc(num, size) _test_calloc(num, size, __FILE__,
__LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Test function overriding realloc which detects buffer overruns
 *        and memoery leaks.
 *
 * @param[in]  ptr   The memory block which should be changed.
 *
 * @param[in]  size  The bytes which should be allocated.
 *
 * @return           The newly allocated memory block, NULL on error.
 */
void *test_realloc(void *ptr, size_t size);
#else
#define test_realloc(ptr, size) _test_realloc(ptr, size, __FILE__,
__LINE__)
#endif

#ifdef DOXYGEN
/**
 * @brief Test function overriding free(3).
 *
 * @param[in]  ptr  The pointer to the memory space to free.
```

```
 *
 * @see free(3).
 */
void test_free(void *ptr);
#else
#define test_free(ptr) _test_free(ptr, __FILE__, __LINE__)
#endif

/* Redirect malloc, calloc and free to the unit test allocators. */
#ifdef UNIT_TESTING
#define malloc test_malloc
#define realloc test_realloc
#define calloc test_calloc
#define free test_free
#endif /* UNIT_TESTING */

/** @} */


/**
 * @defgroup cmocka_mock_assert Standard Assertions
 * @ingroup cmocka
 *
 * How to handle assert(3) of the standard C library.
 *
 * Runtime assert macros like the standard C library's assert() should be
 * redefined in modules being tested to use cmocka's mock_assert()
function.
 * Normally mock_assert() signals a test failure. If a function is called
using
 * the expect_assert_failure() macro, any calls to mock_assert() within
the
 * function will result in the execution of the test. If no calls to
 * mock_assert() occur during the function called via
expect_assert_failure() a
 * test failure is signalled.
 *
 * @{
 */

/**
 * @brief Function to replace assert(3) in tested code.
 *
 * In conjuction with check_assert() it's possible to determine whether
an
 * assert condition has failed without stopping a test.
 *
 * @param[in]  result  The expression to assert.
 *
 * @param[in]  expression  The expression as string.
 *
 * @param[in]  file  The file mock_assert() is called.
 *
 * @param[in]  line  The line mock_assert() is called.
 *
 * @code
 * #ifdef UNIT_TESTING
```

```
 * extern void mock_assert(const int result, const char* const
expression,
 *                         const char * const file, const int line);
 *
 * #undef assert
 * #define assert(expression) \
 *     mock_assert((int)(expression), #expression, __FILE__, __LINE__);
 * #endif
 *
 * void increment_value(int * const value) {
 *     assert(value);
 *     (*value) ++;
 * }
 * @endcode
 *
 * @see assert(3)
 * @see expect_assert_failure
 */
void mock_assert(const int result, const char* const expression,
                 const char * const file, const int line);

#ifdef DOXYGEN
/**
 * @brief Ensure that mock_assert() is called.
 *
 * If mock_assert() is called the assert expression string is returned.
 *
 * @param[in]  fn_call  The function will will call mock_assert().
 *
 * @code
 * #define assert mock_assert
 *
 * void showmessage(const char *message) {
 *   assert(message);
 * }
 *
 * int main(int argc, const char* argv[]) {
 *   expect_assert_failure(show_message(NULL));
 *   printf("succeeded\n");
 *   return 0;
 * }
 * @endcode
 *
 */
void expect_assert_failure(function fn_call);
#else
#define expect_assert_failure(function_call) \
  { \
    const int result = setjmp(global_expect_assert_env); \
    global_expecting_assert = 1; \
    if (result) { \
      print_message("Expected assertion %s occurred\n", \
                    global_last_failed_assert); \
      global_expecting_assert = 0; \
    } else { \
      function_call ; \
      global_expecting_assert = 0; \
      print_error("Expected assert in %s\n", #function_call); \
```

```
      _fail(__FILE__, __LINE__); \
    } \
  }
#endif

/** @} */

/* Function prototype for setup, test and teardown functions. */
typedef void (*UnitTestFunction)(void **state);

/* Function that determines whether a function parameter value is
correct. */
typedef int (*CheckParameterValue)(const LargestIntegralType value,
                                   const LargestIntegralType
check_value_data);

/* Type of the unit test function. */
typedef enum UnitTestFunctionType {
    UNIT_TEST_FUNCTION_TYPE_TEST = 0,
    UNIT_TEST_FUNCTION_TYPE_SETUP,
    UNIT_TEST_FUNCTION_TYPE_TEARDOWN,
    UNIT_TEST_FUNCTION_TYPE_GROUP_SETUP,
    UNIT_TEST_FUNCTION_TYPE_GROUP_TEARDOWN,
} UnitTestFunctionType;

/*
 * Stores a unit test function with its name and type.
 * NOTE: Every setup function must be paired with a teardown function.
It's
 * possible to specify NULL function pointers.
 */
typedef struct UnitTest {
    const char* name;
    UnitTestFunction function;
    UnitTestFunctionType function_type;
} UnitTest;

typedef struct GroupTest {
    UnitTestFunction setup;
    UnitTestFunction teardown;
    const UnitTest *tests;
    const size_t number_of_tests;
} GroupTest;

/* Function prototype for test functions. */
typedef void (*CMUnitTestFunction)(void **state);

/* Function prototype for setup and teardown functions. */
typedef int (*CMFixtureFunction)(void **state);

struct CMUnitTest {
    const char *name;
    CMUnitTestFunction test_func;
    CMFixtureFunction setup_func;
    CMFixtureFunction teardown_func;
    void *initial_state;
};
```

```
/* Location within some source code. */
typedef struct SourceLocation {
    const char* file;
    int line;
} SourceLocation;

/* Event that's called to check a parameter value. */
typedef struct CheckParameterEvent {
    SourceLocation location;
    const char *parameter_name;
    CheckParameterValue check_value;
    LargestIntegralType check_value_data;
} CheckParameterEvent;

/* Used by expect_assert_failure() and mock_assert(). */
extern int global_expecting_assert;
//extern jmp_buf global_expect_assert_env;
extern const char * global_last_failed_assert;

/* Retrieves a value for the given function, as set by "will_return". */
LargestIntegralType _mock(const char * const function, const char* const
file,
                          const int line);

void _expect_function_call(
    const char * const function_name,
    const char * const file,
    const int line,
    const int count);

void _function_called(const char * const function, const char* const
file,
                          const int line);

void _expect_check(
    const char* const function, const char* const parameter,
    const char* const file, const int line,
    const CheckParameterValue check_function,
    const LargestIntegralType check_data, CheckParameterEvent * const
event,
    const int count);

void _expect_in_set(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const LargestIntegralType
values[],
    const size_t number_of_values, const int count);
void _expect_not_in_set(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const LargestIntegralType
values[],
    const size_t number_of_values, const int count);

void _expect_in_range(
    const char* const function, const char* const parameter,
    const char* const file, const int line,
    const LargestIntegralType minimum,
    const LargestIntegralType maximum, const int count);
```

```
void _expect_not_in_range(
    const char* const function, const char* const parameter,
    const char* const file, const int line,
    const LargestIntegralType minimum,
    const LargestIntegralType maximum, const int count);

void _expect_value(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const LargestIntegralType
value,
    const int count);
void _expect_not_value(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const LargestIntegralType
value,
    const int count);

void _expect_string(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const char* string,
    const int count);
void _expect_not_string(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const char* string,
    const int count);

void _expect_memory(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const void* const memory,
    const size_t size, const int count);
void _expect_not_memory(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const void* const memory,
    const size_t size, const int count);

void _expect_any(
    const char* const function, const char* const parameter,
    const char* const file, const int line, const int count);

void _check_expected(
    const char * const function_name, const char * const parameter_name,
    const char* file, const int line, const LargestIntegralType value);

void _will_return(const char * const function_name, const char * const
file,
                  const int line, const LargestIntegralType value,
                  const int count);
void _assert_true(const LargestIntegralType result,
                  const char* const expression,
                  const char * const file, const int line);
void _assert_return_code(const LargestIntegralType result,
                         size_t rlen,
                         const LargestIntegralType error,
                         const char * const expression,
                         const char * const file,
                         const int line);
void _assert_int_equal(
    const LargestIntegralType a, const LargestIntegralType b,
```

```
    const char * const file, const int line);
void _assert_int_not_equal(
    const LargestIntegralType a, const LargestIntegralType b,
    const char * const file, const int line);
void _assert_string_equal(const char * const a, const char * const b,
                          const char * const file, const int line);
void _assert_string_not_equal(const char * const a, const char * const b,
                              const char *file, const int line);
void _assert_memory_equal(const void * const a, const void * const b,
                          const size_t size, const char* const file,
                          const int line);
void _assert_memory_not_equal(const void * const a, const void * const b,
                              const size_t size, const char* const file,
                              const int line);
void _assert_in_range(
    const LargestIntegralType value, const LargestIntegralType minimum,
    const LargestIntegralType maximum, const char* const file, const int
line);
void _assert_not_in_range(
    const LargestIntegralType value, const LargestIntegralType minimum,
    const LargestIntegralType maximum, const char* const file, const int
line);
void _assert_in_set(
    const LargestIntegralType value, const LargestIntegralType values[],
    const size_t number_of_values, const char* const file, const int
line);
void _assert_not_in_set(
    const LargestIntegralType value, const LargestIntegralType values[],
    const size_t number_of_values, const char* const file, const int
line);

void* _test_malloc(const size_t size, const char* file, const int line);
void* _test_realloc(void *ptr, const size_t size, const char* file, const
int line);
void* _test_calloc(const size_t number_of_elements, const size_t size,
                   const char* file, const int line);
void _test_free(void* const ptr, const char* file, const int line);

void _fail(const char * const file, const int line);

void _skip(const char * const file, const int line);

int _run_test(
    const char * const function_name, const UnitTestFunction Function,
    void ** const volatile state, const UnitTestFunctionType
function_type,
    const void* const heap_check_point);
CMOCKA_DEPRECATED int _run_tests(const UnitTest * const tests,
                                 const size_t number_of_tests);
CMOCKA_DEPRECATED int _run_group_tests(const UnitTest * const tests,
                                       const size_t number_of_tests);

/* Test runner */
int _cmocka_run_group_tests(const char *group_name,
                            const struct CMUnitTest * const tests,
                            const size_t num_tests,
                            CMFixtureFunction group_setup,
                            CMFixtureFunction group_teardown);
```

```
/* Standard output and error print methods. */
void print_message(const char* const format, ...)
CMOCKA_PRINTF_ATTRIBUTE(1, 2);
void print_error(const char* const format, ...)
CMOCKA_PRINTF_ATTRIBUTE(1, 2);
void vprint_message(const char* const format, va_list args)
CMOCKA_PRINTF_ATTRIBUTE(1, 0);
void vprint_error(const char* const format, va_list args)
CMOCKA_PRINTF_ATTRIBUTE(1, 0);

enum cm_message_output {
    CM_OUTPUT_STDOUT,
    CM_OUTPUT_SUBUNIT,
    CM_OUTPUT_TAP,
    CM_OUTPUT_XML,
};

/**
 * @brief Function to set the output format for a test.
 *
 * The ouput format for the test can either be set globally using this
 * function or overriden with environment variable CMOCKA_MESSAGE_OUTPUT.
 *
 * The environment variable can be set to either STDOUT, SUBUNIT, TAP or
XML.
 *
 * @param[in] output    The output format to use for the test.
 *
 */
void cmocka_set_message_output(enum cm_message_output output);

/** @} */

#endif /* CMOCKA_H_ */
/**
 * @brief
 *
 * @file dispatcher_task.h
 * @author Gunj Manseta
 * @date 2018-04-26
 */

#ifndef DISPATCHER_H
#define DISPATCHER_H

#include <stdlib.h>
#include <errno.h>
#include <string.h>

#include <mqueue.h>

#include "common_helper.h"
#include "communication_object.h"


/**
 * @brief Get the Handle DispatcherTaskQueue object
```

```
 *
 * @return mqd_t
 */
mqd_t getHandle_DispatcherTaskQueue();


#define POST_MESSAGE_DISPATCHERTASK(p_comm_msg)  \
    do{ \
        __POST_MESSAGE_DISPATCHERTASK(getHandle_DispatcherTaskQueue(),
(p_comm_msg), sizeof(*p_comm_msg), 20); \
    }while(0)

#define POST_MESSAGE_DISPATCHERTASK_EXIT(format, ...)  \
    do{ \
        COMM_MSG_T comm_msg;    \
        /*(strlen(format)>0) ?
snprintf(comm_msg.message,sizeof(comm_msg.message),format,
##__VA_ARGS__):0; */\
        COMM_OBJECT_MSGID(comm_msg,0xFF); \
        COMM_DST_BOARD_ID(comm_msg,BBG_BOARD_ID); \
        __POST_MESSAGE_DISPATCHERTASK(getHandle_DispatcherTaskQueue(),
&comm_msg, sizeof(comm_msg), 20); \
    }while(0)

/**
 * @brief
 *
 * @param queue
 * @param comm_msg
 * @param comm_msg_size
 * @param prio
 */
static inline void __POST_MESSAGE_DISPATCHERTASK(mqd_t queue, const
COMM_MSG_T *comm_msg, size_t comm_msg_size, int prio)
{
    if(-1 == mq_send(queue, (const char*)comm_msg, comm_msg_size, prio))
    {
        LOG_STDOUT(ERROR "DISPATCHER:MQ_SEND:%s\n",strerror(errno));
    }
}

/**
 * @brief
 *
 * @param threadparam
 * @return void*
 */
void* dispatcher_task_callback(void *threadparam);

#endif/**
 * @brief
 *
 * @file BB_Led.h
 * @author Gunj Manseta
 * @date 2018-03-10
 */

#ifndef BB_LED_H
```

```
#define BB_LED_H

typedef enum
{
    LED0,
    LED1,
    LED2,
    LED3
}USER_LED_T;

/**
 * @brief
 *
 * @param lednum
 * @return int
 */
int BB_LedON(USER_LED_T lednum);

/**
 * @brief
 *
 * @param lednum
 * @return int
 */
int BB_LedOFF(USER_LED_T lednum);

/**
 * @brief
 *
 * @param lednum
 * @return int
 */
int BB_LedDefault();

#endif/**
 * @brief
 *
 * @file tmp102_sensor.h
 * @author Gunj Manseta
 * @date 2018-03-13
 */
#ifndef TMP102SENSOR_H
#define TMP102SENSOR_H


#define TMP102_SLAVE_ADDR           (0x48)


/* Register address */
#define TMP102_REG_TEMPERATURE          (0x00)
#define TMP102_REG_CONFIGURATION        (0x01)
#define TMP102_REG_TLOW                 (0x02)
#define TMP102_REG_THIGH                (0x03)



#define TMP102_CONFIG_SD                (1)
#define TMP102_CONFIG_TM                (1<<1)
```

```
#define TMP102_CONFIG_POL               (1<<2)
#define TMP102_CONFIG_EM                (1<<12)
#define TMP102_CONFIG_AL                (1<<13)
#define TMP102_CONFIG_CR(x)             (x<<14)

#define TMP102_CONFIG_FAULTBITS                 (3<<3)
/*generates alert after 4 consecutive faults*/
#define TMP102_CONFIG_ONESHOT_CR                (1<7)               /*saves
power between conversions when 1*/

typedef enum temperature_unit
{
    CELCIUS = 0,
    FAHREN,
    KELVIN

}TEMPERATURE_UNIT_T;

typedef struct
{
    ///* (D)0 = Continuous conversion; 1 = Can sleep*/
    uint16_t SD_MODE:1;
    ///* (D)0 = Comparatore mode; 1 = Interrupt mode*/
    uint16_t TM_MODE:1;
    ///* (D)0 = ALERT pin becomes active low; 1 = ALERT pin becomes
active high and the state of the ALERT pin is inverted. */
    uint16_t POL:1;
    ///* (D)0 = 1Fault; 1= 2Faults; 3 = 4Faults; 4=6Faults */
    uint16_t FAULT:2;
    const uint16_t RES0:2;
    ///* (D)0 =  During the conversion, the OS bit reads '0'; 1 = writing
a 1 to the OS bit starts a single temperature conversion */
    uint16_t OS:1;
    const uint16_t RES1:4;
    ///* (D)0 = Normal mode(12bit); 1= Extended mode(13 bit) */
    uint16_t EM_MODE:1;
    ///*  Reads the AL bit*/
    const uint16_t RO_AL_MODE:1;
    ///* 0 = 0.25Hz ; 1 = 1Hz ; (D)2 = 4Hz ; 3 = 8Hz */
    uint16_t CR:2;

}TMP102_CONFIG_REG_SETTINGS_T;

#define TMP102_CONFIG_DEFAULT_ASSIGN \
{\
    .SD_MODE = 0,\
    .TM_MODE = 0,\
    .POL     = 0,\
    .OS      = 0,\
    .EM_MODE = 0,\
    .CR      = 2\
}

extern const TMP102_CONFIG_REG_SETTINGS_T TMP102_CONFIG_DEFAULT;

int TMP102_setMode(TMP102_CONFIG_REG_SETTINGS_T mode);

/**
```

```
 * @brief Brings back to default
 *
 * @return int
 */
int TMP102_setmode_allDefault();

/**
 * @brief Gives a memdump of 4 len.
 *  **IMP** must free the address using return pointer
 * @return uint8_t*
 */
uint16_t* TMP102_memDump();

/**
 * @brief Abstracted macros for different units
 *
 */
#define TMP102_getTemp_Celcius(p_temp)  TMP102_getTemp(p_temp, CELCIUS)
#define TMP102_getTemp_Kelvin(p_temp)   TMP102_getTemp(p_temp, KELVIN)
#define TMP102_getTemp_Fahren(p_temp)   TMP102_getTemp(p_temp, FAHREN)


/**
 * @brief Gets the temperature value
 *
 * @param temp
 * @param unit
 * @return int
 */
int TMP102_getTemp(float *temp, TEMPERATURE_UNIT_T unit);

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_SD_PowerSaving();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_SD_Continuous_default();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_TM_ComparatorMode_default();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_TM_InterruptMode();
```

```
/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_ALERT_ActiveLow_default();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_ALERT_ActiveHigh();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_EM_NormalMode_default();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_EM_ExtendedMode();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_CR_250mHZ();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_CR_1HZ();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_CR_4HZ_default();

/**
 * @brief
 *
 * @return int
 */
int TMP102_setMode_CR_8HZ();

/**
```

```
 * @brief
 *
 * @param al_bit
 * @return int
 */
int TMP102_readMode_ALERT(uint8_t *al_bit);

/**
 * @brief
 *
 * @param tlow_C
 * @return int
 */
int TMP102_write_Tlow(float tlow_C);

/**
 * @brief
 *
 * @param thigh_C
 * @return int
 */
int TMP102_write_Thigh(float thigh_C);

/**
 * @brief
 *
 * @param tlow_C
 * @return int
 */
int TMP102_read_Tlow(float *tlow_C);

/**
 * @brief
 *
 * @param thigh_C
 * @return int
 */
int TMP102_read_Thigh(float *thigh_C);


#endif/**
 * @brief
 *
 * @file my_uart.h
 * @author Gunj Manseta
 * @date 2018-04-23
 */

#ifndef MYUART_H
#define MYUART_H

#include <stdlib.h>

typedef enum
{
    COM_PORT1 = 1,  //"/dev/ttyS1"
    COM_PORT2,      //"/dev/ttyS2"
    COM_PORT3,      //"/dev/ttyS3"
```

```
    COM_PORT4,      //"/dev/ttyS4"
}COM_PORT;

typedef int UART_FD_T;

/**
 * @brief
 *
 * @return UART_FD_T
 */
UART_FD_T UART_Open(COM_PORT com_port);

/**
 * @brief
 *
 * @param fd
 */
void UART_Close(UART_FD_T fd);


/**
 * @brief
 *
 * @param c
 * @return int32_t
 */
int32_t UART_putchar(char c);


/**
 * @brief
 *
 * @param object
 * @param len
 * @return int32_t
 */
int32_t UART_putRAW(void *object, size_t len);

/**
 * @brief
 *
 * @param str
 * @return int32_t
 */
int32_t UART_putstr(const char* str);

/**
 * @brief
 *
 * @param object
 * @param len
 * @return int32_t
 */
int32_t UART_read(void *object, size_t len);

/**
 * @brief
 *
```

```
 * @param time_ms
 * @return int32_t
 */
int32_t UART_dataAvailable(uint32_t time_ms);

/**
 * @brief
 *
 */
void UART_flush();


#endif/**
* @file - spi.h
* @brief - Header file for the library functions for SPI
*
* @author Gunj University of Colorado Boulder
* @date - 19th April 2018
**/

#if 1
#ifndef _SPI_H_
#define _SPI_H_

#include <stdbool.h>
#include <stdint.h>

#include "mraa/spi.h"

#include "my_uart.h"

#define SPI_1MZ 1000000
#define SPI_2MZ 2000000

#define NOP 0xFF

/**
* @brief - Enum to allow flexibility of selection between SPI0 and SPI1
**/
typedef enum{
      SPI_0,
      SPI_1,
      SPI_2,
      SPI_3
}SPI_t;

#define NUM_SPI_BUS 4
typedef mraa_spi_context SPI_Type;

SPI_Type SPI[NUM_SPI_BUS];

/**
* @brief - Initialize the GPIO pins associated with SPI
* Configure SPI in 3 wire mode and use a GPIO pin for chip select
* @param - spi SPI_t
* @return void
**/
void SPI_GPIO_init(SPI_t spi);
```

```
/**
* @brief - Enable the clock gate control for SPI
* @param - spi SPI_t
* @return void
**/
static inline void SPI_clock_init(SPI_t spi, uint32_t g_sysclock)
{

}



/**
* @brief - Perform the initialization routine for the SPI module
* @param - spi SPI_t
* @return void
**/
SPI_t SPI_init(SPI_t spi);

/**
* @brief - Disable the initialized SPI module
* @return void
**/
SPI_t SPI_disable(SPI_t spi);

/**
* @brief - Blocks until SPI transmit buffer has completed transmitting
* @param - spi SPI_t
* @return void
**/
static inline void SPI_flush(SPI_t spi)
{

}

static inline void SPI_flushRXFIFO(SPI_t spi)
{
    /* Check if it doesnt get an infinite loop */
    while(mraa_spi_write(SPI[spi], NOP));
}


/**
* @brief - Read a single byte from the SPI bus
* @param - spi SPI_t
* @return uint8_t
**/
static inline int8_t SPI_read_byte(SPI_t spi)
{
    return mraa_spi_write(SPI[spi], NOP);
}

/**
* @brief - Read a single byte from the SPI bus without waiting
* @param - spi SPI_t
* @return uint8_t
```

```
**/
static inline int8_t SPI_read_byte_NonBlocking(SPI_t spi)
{
    return mraa_spi_write(SPI[spi], NOP);
}

/**
* @brief - Write a single byte on to the SPI bus
* @param - spi SPI_t
* @param - byte uint8_t
* @return uint8_t
**/
static inline int8_t SPI_write_byte(SPI_t spi, uint8_t byte)
{
    return mraa_spi_write(SPI[spi],byte);
}

/**
* @brief - Write a single byte on to the SPI bus without blocking
* @param - spi SPI_t
* @param - byte uint8_t
* @return void
**/
static inline void SPI_write_byte_NonBlocking(SPI_t spi, uint8_t byte)
{
    mraa_spi_write(SPI[spi],byte);
}

/**
* @brief - Send a packet on to the SPI bus
* Send multiple bytes given a pointer to an array and the number of bytes
to be sent
* @param - spi SPI_t
* @param - p uint8_t
* @param - length size_t
* @return void
**/
int8_t SPI_write_packet(SPI_t spi, uint8_t* p, size_t length);

/**
* @brief - Read a packet from the SPI bus
* Read multiple bytes given a pointer to an array for storage and the
number of bytes to be read
* @param - spi SPI_t
* @param - p uint8_t *
* @param - length size_t
* @return void
**/
int32_t SPI_read_packet(SPI_t spi, uint8_t* p, size_t length);


#endif /* SOURCES_SPI0_H_ */

#endif/**
 * @brief
 *
 * @file my_i2c.h
 * @author Gunj Manseta
```

```
 * @date 2018-03-13
 */
#ifndef MYI2C_H
#define MYI2C_H

#include <pthread.h>
#include "mraa/i2c.h"

#define BB_I2C_BUS_2  (2)

/**
 * @brief This is the handle for I2C mster and each master should have
only one handle
 *
 */
typedef struct i2c_handle
{
    /* This context is a typedef'ed pointer within */
    mraa_i2c_context i2c_context;
    pthread_spinlock_t handle_lock;

}I2C_MASTER_HANDLE_T;

/**
 * @brief Get the MasterI2C handle object
 *
 * @return I2C_MASTER_HANDLE_T *
 */
I2C_MASTER_HANDLE_T* getMasterI2C_handle();

/**
 * @brief Prints the error code string to stdout
 *
 * @param errorCode
 */
void printErrorCode(int errorCode);

/**
 * @brief Inits the I2C master handle
 * There is an internal state of the context which is maintained which
gets updated with every init call
 * Internal context goes to NULL is error in init
 * This context points to the new handle that is passed as the parameter
 * @param handle
 * @return int SUCCESS=0 and ERROR =-1
 */
int I2Cmaster_Init(I2C_MASTER_HANDLE_T *handle);

/**
 * @brief
 *
 * @param handle
 * @return int
 */
int I2Cmaster_Destroy(I2C_MASTER_HANDLE_T *handle);

/**
 * @brief
```

```
 *
 * @param slave_addr
 * @param reg_addr
 * @param data
 * @return int
 */
int I2Cmaster_read_byte(uint8_t slave_addr, uint8_t reg_addr, uint8_t
*data);

/**
 * @brief
 *
 * @param slave_addr
 * @param reg_addr
 * @param data
 * @param len
 * @return int
 */
int I2Cmaster_read_bytes(uint8_t slave_addr, uint8_t reg_addr, uint8_t
*data, size_t len);


/**
 * @brief Writes a byte/pointer register to the slave
 *
 * @param slave_addr
 * @param reg_addr
 * @return int
 */
int I2Cmaster_write(uint8_t slave_addr, uint8_t reg_addr);

/**
 * @brief
 *
 * @param slave_addr
 * @param reg_addr
 * @param data
 * @return int
 */
int I2Cmaster_write_byte(uint8_t slave_addr, uint8_t reg_addr, uint8_t
data);

/**
 * @brief
 *
 * @param slave_addr
 * @param reg_addr
 * @param data
 * @param len
 * @return int
 */
int I2Cmaster_write_bytes(uint8_t slave_addr, uint8_t reg_addr, uint8_t
*data, size_t len);

/**
 * @brief
 *
 * @param slave_addr
```

```
 * @param reg_addr
 * @param data
 * @param lsb_first
 * @return int
 */
int I2Cmaster_write_word(uint8_t slave_addr, uint8_t reg_addr, uint16_t
data, uint8_t lsb_first);



#endif/**
 * @brief
 *
 * @file apds9301_sensor.h
 * @author Gunj Manseta
 * @date 2018-03-13
 */


#ifndef APDS9301SENSOR_H
#define APDS9301SENSOR_H

#include <stdint.h>

#define APDS9301_SLAVE_ADDR     (0x39)

#define APDS9301_CMD_REG        (0x80)
#define APDS9301_CMD_WORD_EN    (1<<5)
#define APDS9301_CMD_INT_CLEAR  (1<6)

/* REGISTERS */
#define APDS9301_CTRL_REG           (0x00) | APDS9301_CMD_REG
#define APDS9301_TIMING_REG         (0x01) | APDS9301_CMD_REG
#define APDS9301_ID_REG             (0x0A) | APDS9301_CMD_REG
#define APDS9301_INT_CTRL_REG       (0x06) | APDS9301_CMD_REG
#define APDS9301_CH0_DATALOW        (0x0C) | APDS9301_CMD_REG
#define APDS9301_CH0_DATAHIGH       (0x0D) | APDS9301_CMD_REG
#define APDS9301_CH1_DATALOW        (0x0E) | APDS9301_CMD_REG
#define APDS9301_CH1_DATAHIGH       (0x0F) | APDS9301_CMD_REG
#define APDS9301_INT_TH_LL_REG      (0x02) | APDS9301_CMD_REG
#define APDS9301_INT_TH_LH_REG      (0x03) | APDS9301_CMD_REG
#define APDS9301_INT_TH_HL_REG      (0x04) | APDS9301_CMD_REG
#define APDS9301_INT_TH_HH_REG      (0x05) | APDS9301_CMD_REG


/* Bit fields in Registers */
#define APDS9301_CTRL_POWERON       (0x03)
#define APDS9301_CTRL_POWEROFF      (0x00)
#define APDS9301_INTCTRL_IEN        (1<<4)
#define APDS9301_TIMING_GAIN        (1<<4)
#define APDS9301_TIMING_INTEG(x)    (x)
#define APDS9301_TIMING_MANUAL(x)   (x<<3)


#define  APDS9301_mode_interruptEnable()
APDS9301_mode_interrupt(1)
#define  APDS9301_mode_interruptDisable_default()
APDS9301_mode_interrupt(0)
```

```
#define APDS9301_mode_integrationTime0()
APDS9301_mode_integrationTime(0)
#define APDS9301_mode_integrationTime1()
APDS9301_mode_integrationTime(1)
#define APDS9301_mode_integrationTime2_default()
APDS9301_mode_integrationTime(2)
#define APDS9301_mode_integrationTime3()
APDS9301_mode_integrationTime(3)

#define APDS9301_mode_manualcontrolON()
APDS9301_mode_manualcontrol(1)
#define APDS9301_mode_manualcontrolOFF_default()
APDS9301_mode_manualcontrol(0)


/**
 * @brief Sets back the default configration of the sensor
 *
 * @return int
 */
int APDS9301__setmode_allDefault();

/**
 * @brief Gives a memdump of 15 len.
 *  **IMP** must free the address using return pointer
 * @return uint8_t*
 */
uint8_t* APDS9301_memDump();

/**
 * @brief
 *
 * @param thlow
 * @return int
 */
int APDS9301_write_ThLow(uint16_t thlow);

/**
 * @brief
 *
 * @param thhigh
 * @return int
 */
int APDS9301_write_ThHigh(uint16_t thhigh);

/**
 * @brief
 *
 * @param thlow
 * @return int
 */
int APDS9301_read_ThLow(uint16_t *thlow);

/**
 * @brief
 *
 * @param thhigh
```

```
 * @return int
 */
int APDS9301_read_ThHigh(uint16_t *thhigh);

/**
 * @brief
 *
 * @param ctrl_reg
 * @return int
 */
int APDS9301_readControlReg(uint8_t *ctrl_reg);

/**
 * @brief
 *
 * @return int
 */
int APDS9301_mode_highGain();

/**
 * @brief
 *
 * @return int
 */
int APDS9301_mode_lowGain_default();

/**
 * @brief
 *
 * @param on
 * @return int
 */
int APDS9301_mode_manualcontrol(uint8_t on);

/**
 * @brief
 *
 * @param x
 * @return int
 */
int APDS9301_mode_integrationTime(uint8_t x);

/**
 * @brief
 *
 * @param enable
 * @return int
 */
int APDS9301_mode_interrupt(uint8_t enable);

/**
 * @brief
 *
 * @return int
 */
int APDS9301_clearPendingInterrupt();
```

```
/**
 * @brief
 *
 * @return int
 */
int APDS9301_poweron();

/**
 * @brief
 *
 * @return int
 */
int APDS9301_powerdown();

/**
 * @brief
 *
 * @param id
 * @return int
 */
int APDS9301_readID(uint8_t *id);

/**
 * @brief
 *
 * @param ch0_data
 * @return int
 */
int APDS9301_readCh0(uint16_t *ch0_data);

/**
 * @brief
 *
 * @param ch1_data
 * @return int
 */
int APDS9301_readCh1(uint16_t *ch1_data);

/**
 * @brief
 *
 * @return float
 */
float APDS9301_getLux();

/**
 * @brief
 *
 * @return int
 */
int APDS9301_test();


#endif/**
 * @brief Test for the APDS9301 sensor
 *
 * @file apds9301_testmain.c
 * @author Gunj Manseta
```

```
 * @date 2018-03-14
 */

#include "my_i2c.h"
#include "apds9301_sensor.h"
#include <unistd.h>

int main()
{
    I2C_MASTER_HANDLE_T i2c;
    int ret = 0;
    if(ret = I2Cmaster_Init(&i2c) !=0)
    {
        printErrorCode(ret);
        printf("[ERROR] I2C Master init failed\n");
    }

    ret = APDS9301_poweron();
    if(ret == 0) printf("Sensor ON\n");
    uint8_t sensor_id = 0x50;
    uint8_t data = 0;
    ret  = APDS9301_readControlReg(&data);
    if(ret == 0) printf("CTRL REG: %x\n",data);

    uint16_t tlow;
    ret = APDS9301_read_ThLow(&tlow);
    if(ret == 0) printf("READ TLOW 0x%x\n",tlow);

    tlow = 0xBB11;
    ret = APDS9301_write_ThLow(tlow);
    if(ret == 0) printf("WRITE TLOW 0x%x\n",tlow);

    tlow = 0xaaaa;
    ret = APDS9301_read_ThLow(&tlow);
    if(ret == 0) printf("READ TLOW 0x%x\n",tlow);

    ret = APDS9301_mode_highGain();
    if(ret != 0) printf("ERROR\n");

    uint8_t *memdump = APDS9301_memDump();
    printf("----SENSOR DUMP-----\n");
    for(uint8_t i = 0; i < 15; i++)
        printf("%02dh : 0x%x\n",i,memdump[i]);
    free(memdump);
    printf("---------\n");

    ret = APDS9301_mode_lowGain_default();
    if(ret != 0) printf("ERROR\n");

    ret = APDS9301_readID(&data);
    if(ret == 0) printf("expected: %x ID: %x\n",sensor_id, data);

    while(1)
    {
        float lux = APDS9301_getLux();
        if(lux < 0) printf("Error. Lux is negative\n");
        else    printf("Lux: %f\n",lux);
        sleep(2);
```

```
    }

    if(ret = I2Cmaster_Destroy(&i2c) !=0)
    {
        printErrorCode(ret);
        printf("[ERROR] I2C Master destroy failed\n");
    }

}
/**
 * @brief Test for the APDS9301 sensor
 *
 * @file apds9301_testmain.c
 * @author Gunj Manseta
 * @date 2018-03-14
 */

#include "my_i2c.h"
#include "apds9301_sensor.h"
#include <unistd.h>
#include "cmocka.h"

void testAPDS9301(void **state)
{
    I2C_MASTER_HANDLE_T i2c;
    int ret = 0;
    ret = I2Cmaster_Init(&i2c);
    assert_int_equal(ret, 0);
    assert_non_null((void*)getMasterI2C_handle());
    assert_ptr_equal(&i2c,getMasterI2C_handle());

    ret = APDS9301_poweron();
    assert_int_equal(ret, 0);

    ret = APDS9301_test();
    assert_int_equal(ret, 0);

    uint8_t data = 0;
    ret  = APDS9301_readControlReg(&data);
    assert_int_equal(ret, 0);
    assert_int_equal((data & 0x3), 0x03);


    uint16_t tlow = 0xBB11;
    ret = APDS9301_write_ThLow(tlow);
    assert_int_equal(ret, 0);

    tlow = 0;
    ret = APDS9301_read_ThLow(&tlow);
    assert_int_equal(ret, 0);
    assert_int_equal(tlow, 0xBB11);

    uint16_t thigh = 0xA5A5;
    ret = APDS9301_write_ThHigh(thigh);
    assert_int_equal(ret, 0);

    thigh = 0 ;
    ret = APDS9301_read_ThHigh(&thigh);
```

```
assert_int_equal(ret, 0);
assert_int_equal(thigh, 0xA5A5);


ret = APDS9301_mode_highGain();
assert_int_equal(ret, 0);

ret = APDS9301_mode_integrationTime3();
assert_int_equal(ret, 0);

ret = APDS9301_mode_interruptEnable();
assert_int_equal(ret, 0);

ret = APDS9301_mode_manualcontrolON();
assert_int_equal(ret, 0);

uint8_t *memdump = APDS9301_memDump();

assert_non_null(memdump);

/* Power up bits */
assert_int_equal(memdump[0] & 0x3, 0x3);

/* Timing register */
assert_int_equal(memdump[1] & 0x1B, 0x1B);

/* Interrupt control reg */
assert_int_equal(memdump[6] & 0x3F, 0x10);

free(memdump);

// ret = APDS9301_mode_lowGain_default();
// assert_int_equal(ret, 0);

// ret = APDS9301_mode_integrationTime2_default();
// assert_int_equal(ret, 0);

// ret = APDS9301_mode_interruptDisable_default();
// assert_int_equal(ret, 0);

// ret = APDS9301_mode_manualcontrolOFF_default();
// assert_int_equal(ret, 0);

ret = APDS9301__setmode_allDefault();
assert_int_equal(ret, 0);

ret = APDS9301_readID(&data);
assert_int_equal(data&0xF0, 0x50);

int i = 0;
while(i<2)
{
    float lux = APDS9301_getLux();
    assert_int_not_equal(lux, -1);
    assert_in_range(lux,0,100);

    i++;
}
```

```
    ret = APDS9301_powerdown();
    assert_int_equal(ret, 0);

    ret = APDS9301_readControlReg(&data);
    assert_int_equal(ret, 0);
    assert_int_equal((data & 0x3), 0);

    ret = I2Cmaster_Destroy(&i2c);
    assert_int_equal(ret, 0);
    assert_null((void*)getMasterI2C_handle());

}


int main()
{
    const struct CMUnitTest tests[] = {

    cmocka_unit_test(testAPDS9301)

      };

      return cmocka_run_group_tests(tests, NULL, NULL);

}/**
 * @brief
 *
 * @file LED_test.c
 * @author Gunj Manseta
 * @date 2018-03-10
 */

#include <stdio.h>
#include <unistd.h>
#include "BB_Led.h"


int main()
{
    if(!BB_LedON(1))
        printf("LED ON\n");
    sleep(5);
    if(!BB_LedOFF(1))
        printf("LED OFF\n");
    return 0;
}#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <stdio.h>
#include <arpa/inet.h>
#include <signal.h>

#include "sensor_common_object.h"

//#define PORT 3000
```

```
//#define IP "127.0.0.1"
//#define IP "192.168.1.238"
#define LOG(format, ...) printf(format, ##__VA_ARGS__)

int client_socket = 0;

void handler(int sig)
{
    close(client_socket);
    LOG("SINGAL - Socket Closed\n");
}

void printResponse(REMOTE_RESPONSE_T rsp);
void printfMENU();

int main()
{
    signal(SIGTERM, handler);
    signal(SIGTSTP, handler);
    struct sockaddr_in addr, server_addr = {0};
    uint16_t PORT = 3000;
    char IP[20] = "192.168.7.2";


    REMOTE_REQUEST_T req = {0};
    REMOTE_RESPONSE_T rsp = {0};

    if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        LOG("[ERROR] Socket creation\n");
        return -1;
    }

    LOG("[INFO] Socket Created\n");

    //memset(&server_addr, 0, sizeof(server_addr));

    server_addr.sin_family = AF_INET;

    LOG("***CLIENT APPLICATION***\n");
    LOG("Default IP:%s PORT%u\n",IP,PORT);
    LOG("Enter new IP and Port?(y/n)->");
    char ans;
    scanf(" %c",&ans);
    if(ans == 'y' || ans == 'Y')
    {
        LOG("Enter Port number ->");
        scanf(" %hu",&PORT);
        LOG("Enter IP addr ->");
        scanf("%s",IP);
        //fgets(IP, 20, stdin);
    }

    server_addr.sin_port = htons(PORT);

    /* We need this to convert the IP ADDR in proper format */
    if(inet_pton(AF_INET, IP, &server_addr.sin_addr)<=0)
    {
```

```
        LOG("[ERROR] Invalid address\n");
        return -1;
    }

    LOG("Continue?(y/n)->");
    scanf(" %c",&ans);
    if(ans == 'n' || ans == 'N')
        exit(0);

    if (connect(client_socket, (struct sockaddr *)&server_addr,
sizeof(server_addr)) < 0)
    {
        LOG("[ERROR] Connection Failed \n");
        return -1;
    }
    int i = 0, nbytes;

    int continue_flag = 1;
    do{
        printfMENU();
        LOG("\nChoice --> ");
        fflush(stdin);
        scanf(" %c",&ans);
        if(ans > '9' || ans < '1')
        {
            LOG("INVALID OPTION");
            ans = 0;
            continue;
        }

        //option = (ans - 48);
        //printf("option: %d\n",option);

        //req.request_id = GET_TEMP_C + i%7;
        req.request_id = ans-48-1;
        /*Sending the payload */
        nbytes = send(client_socket , (char*)&req , sizeof(req), 0 );
        if(nbytes < sizeof(req))
        {
            LOG("[ERROR] Cannot send complete data\n");
            return 1;
        }

        //LOG("[INFO] Number of bytes sent: %d\n",nbytes);

        nbytes=0;
        do{
            nbytes = recv(client_socket, (((char*)&(rsp))+nbytes),
sizeof(rsp), 0);
        }while(nbytes < sizeof(rsp) && nbytes != -1);

        //LOG("[INFO] Receivced bytes: %d\n",nbytes);

        LOG("\n***SERVER RESPONSE***");
        printResponse(rsp);
        LOG("********************\n");

        if(ans == '8' || ans == '9')
```

```
        {
            continue_flag = 0;
        }
        i++;
    }while(continue_flag);

    close(client_socket);

    LOG("[INFO] Connection closed\n");
    return 0;
}


void printfMENU()
{
    LOG("\n***********MENU***********");
    LOG("\nSelect from the below options");
    LOG("\n1. Get Temperature Value in C");
    LOG("\n2. Get Temperature Value in F");
    LOG("\n3. Get Temperature Value in K");
    LOG("\n4. Get LUX Value");
    LOG("\n5. Check if Day/Night");
    LOG("\n6. Get Distance in cm");
    LOG("\n7. Get Distance in m");
    LOG("\n8. Close the connection and exit");
    LOG("\n9. Close the remote app");
    LOG("\n**************************");
}

void printResponse(REMOTE_RESPONSE_T rsp)
{
    LOG("\n");
    switch(rsp.rsp_id)
    {
        case(GET_FUNC):
            LOG("%s",rsp.metadata);
            break;
        case(GET_TEMP_C):
            LOG("degree C : %0.3f",rsp.data.floatingData);
            break;
        case(GET_TEMP_F):
            LOG("degree F : %0.3f",rsp.data.floatingData);
            break;
        case(GET_TEMP_K):
            LOG("degree K : %0.3f",rsp.data.floatingData);
            break;
        case(GET_LUX):
            LOG("LUX : %0.3f",rsp.data.floatingData);
            break;
        case(GET_DAY_NIGHT):
            LOG("It is %s now",((rsp.data.isNight == 0) ? "DAY" :
"NIGHT"));
            break;
        case(GET_DISTANCE_CM):
            LOG("Recent distance is %.2f cm",rsp.data.floatingData);
            break;
        case(GET_DISTANCE_M):
            LOG("Recent distance is %.2f m",rsp.data.floatingData);
```

```
            break;
        // case(CONN_CLOSE_RSP):
        //     break;
        default:
            break;
    }
    LOG("\n");
}/**
 * @brief Test for tmp102 sensor
 *
 * @file tmp102_testmain.c
 * @author Gunj Manseta
 * @date 2018-03-14
 */

#include "my_i2c.h"
#include "tmp102_sensor.h"
#include <unistd.h>
#include "cmocka.h"

I2C_MASTER_HANDLE_T i2c;



static void testTMP102(void **state)
{
    int ret = I2Cmaster_Init(&i2c);
    assert_int_equal(ret, 0);
    assert_non_null((void*)getMasterI2C_handle());
    assert_ptr_equal(&i2c,getMasterI2C_handle());

    uint16_t *memdump = TMP102_memDump();
    assert_non_null(memdump);
    // printf("----1.SENSOR DUMP-----\n");
    // for(uint8_t i = 0; i < 4; i++)
    // {
    //     (i == 1) ? assert_int_equal(memdump[i], 0x60a0): 0;
    //     (i == 2) ? assert_int_equal(memdump[i], 0x4b00): 0;
    //     (i == 3) ? assert_int_equal(memdump[i], 0x5000): 0;
    //     printf("%02dh : 0x%x\n",i,memdump[i]);
    // }
    // printf("---------\n");

    assert_int_equal(memdump[1], 0x60a0);
    assert_int_equal(memdump[2], 0x4b00);
    assert_int_equal(memdump[3], 0x5000);

    free(memdump);

    ret = TMP102_setMode_ALERT_ActiveHigh();
    assert_int_equal(ret, 0);
    ret = TMP102_setMode_CR_8HZ();
    assert_int_equal(ret, 0);
    ret = TMP102_setMode_SD_PowerSaving();
    assert_int_equal(ret, 0);
    ret = TMP102_setMode_TM_InterruptMode();
    assert_int_equal(ret, 0);
    ret = TMP102_setMode_EM_ExtendedMode();
```

```
assert_int_equal(ret, 0);


memdump = TMP102_memDump();
assert_non_null(memdump);
// printf("----2.SENSOR DUMP-----\n");
// for(uint8_t i = 0; i < 4; i++)
// {
//     (i == 1) ? assert_int_equal(memdump[i], 0x67d0): 0;
//     (i == 2) ? assert_int_equal(memdump[i], 0x4b00): 0;
//     (i == 3) ? assert_int_equal(memdump[i], 0x5000): 0;
//     printf("%02dh : 0x%x\n",i,memdump[i]);
// }
// printf("---------\n");

assert_int_equal(memdump[1], 0x67d0);
assert_int_equal(memdump[2], 0x4b00);
assert_int_equal(memdump[3], 0x5000);

free(memdump);



ret = TMP102_setmode_allDefault();
assert_int_equal(ret, 0);

memdump = TMP102_memDump();
// printf("----3.SENSOR DUMP-----\n");
// for(uint8_t i = 0; i < 4; i++)
// {
//     (i == 1) ? assert_int_equal(memdump[i], 0x60a0): 0;
//     (i == 2) ? assert_int_equal(memdump[i], 0x4b00): 0;
//     (i == 3) ? assert_int_equal(memdump[i], 0x5000): 0;

//     printf("%02dh : 0x%x\n",i,memdump[i]);
// }
// printf("---------\n");

assert_int_equal(memdump[1], 0x60a0);
assert_int_equal(memdump[2], 0x4b00);
assert_int_equal(memdump[3], 0x5000);

/* Checking the Tlow = 75deg C and Thigh 80deg C */
float temp = (float)(memdump[2]>>4) * 0.0625;
assert_int_equal(temp,75.0);

temp = (float)(memdump[3]>>4) * 0.0625;
assert_int_equal(temp,80.0);

free(memdump);

float temperature = 0.0, celcius = 0.0f, dummy = 0.0;

int i = 0;

printf("\n---------TEMPERATURE VALUES--------------\n");
while(i<3)
{
```

```
            int ret = TMP102_getTemp_Celcius(&temperature);
        assert_int_equal(ret, 0);
            if(ret == 0) printf("C Temp: %.03f\n",temperature);
        celcius = temperature;

            ret = TMP102_getTemp_Fahren(&temperature);
        assert_int_equal(ret, 0);
        dummy = (celcius*1.8) + 32;
        assert_true(temperature == dummy);
            if(ret == 0) printf("F Temp: %.03f\n",temperature);

            ret = TMP102_getTemp_Kelvin(&temperature);
        assert_int_equal(ret, 0);
        dummy = celcius + 273.15;
        assert_true(temperature == dummy);
            if(ret == 0) printf("K Temp: %.03f\n",temperature);

        i++;
        sleep(1);
    }
    printf("----------------------------------------\n");


    ret = I2Cmaster_Destroy(&i2c);
    assert_int_equal(ret, 0);
    assert_null((void*)getMasterI2C_handle());

}


int main()
{
    const struct CMUnitTest tests[] = {

    cmocka_unit_test(testTMP102)

      };

      return cmocka_run_group_tests(tests, NULL, NULL);

}/**
 * @brief
 *
 * @file logger_task.c
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>

#include "main_task.h"
#include "logger_task.h"
#include "error_data.h"
```

```
#include "readConfiguration.h"

#define LOG_DIR     "./log/"
#define __LOG_PATH(x) LOG_DIR ## x
#define LOG_PATH(x) __LOG_PATH(x)

#define MQ_LOGGERTASK_NAME "/loggertask_queue"


/**
 * @brief USe it carefully as there is not NULL checking of the file
stream provided
 *
 */
#define LT_LOG(fp,format, ...)
do{fprintf(fp,"[PID:%d][TID:%ld]",getpid(),syscall(SYS_gettid));
fprintf(fp,format, ##__VA_ARGS__); fflush(fp);}while(0)

#define LT_LOG_COMM(fp,recv_comm_msg)    \
            ({LT_LOG(fp,INFO "\n*******\
            \nSRCID:%u, SRC_BRDID:%u, DST_ID:%u, DST_BRDID:%u MSGID:%u\
            \nSensorVal: %.2f MSG:%s\
            \nChecksum:%u\n********\n",\
            recv_comm_msg.src_id, recv_comm_msg.src_brd_id,
recv_comm_msg.dst_id,recv_comm_msg.dst_brd_id,recv_comm_msg.msg_id,recv_c
omm_msg.data.distance_cm,recv_comm_msg.message,recv_comm_msg.checksum);})

/* Keeping the log level to the higest level to log everything.
    Should be configure at compile time using compile time switch
 */
LOG_LEVEL_T g_loglevel = LOG_ALL;

static mqd_t loggertask_q;

mqd_t getHandle_LoggerTaskQueue()
{
    return loggertask_q;
}

FILE* logger_task_file_init(const char *logFileName)
{
    if(NULL == logFileName)
        return NULL;
    FILE *fp = fopen(logFileName,"r+");
    /* check if the file already exists then close it and save it as
old_log */
    if(fp)
    {
        fclose(fp);
        char newFilename[40] = {0};

snprintf(newFilename,sizeof(newFilename),"%u_%s",(unsigned)time(NULL),log
FileName);
        int ret = rename(logFileName, newFilename);
        if(ret)
        {
            LOG_STDOUT(ERROR "Cannot backup old log file\n");
        }
```

```
    }
    fp = fopen(logFileName,"w+");
    if(NULL == fp)
    {
        LOG_STDOUT(INFO "Log file created\n");
    }
    return fp;
}

int logger_task_queue_init()
{
    struct mq_attr loggertaskQ_attr = {
        .mq_msgsize = sizeof(LOGGERTASKQ_MSG_T),
        .mq_maxmsg = 128,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_LOGGERTASK_NAME);
    loggertask_q = mq_open(MQ_LOGGERTASK_NAME, O_CREAT | O_RDWR, 0666,
&loggertaskQ_attr);

    return loggertask_q;;
}

void logger_task_processMsg(FILE *fp)
{
    int ret,prio;
    LOGGERTASKQ_MSG_T queueData = {0};
    DEFINE_MAINTASK_STRUCT(maintaskRsp,MT_MSG_STATUS_RSP,LOGGER_TASK_ID);
    //struct timespec recv_timeout = {0};
    uint8_t continue_flag= 1;
    while(continue_flag)
    {
        memset(&queueData,0,sizeof(queueData));
        // clock_gettime(CLOCK_REALTIME, &recv_timeout);
        // recv_timeout.tv_sec += 3;
        // ret =
mq_timedreceive(loggertask_q,(char*)&(queueData),sizeof(queueData),&prio,
&recv_timeout);
        ret =
mq_receive(loggertask_q,(char*)&(queueData),sizeof(queueData),&prio);
        // if(ERR == ret && ETIMEDOUT == errno)
        // {
        //     //LOG_STDOUT(ERROR "MQ_RECV
TIMEOUT:%s\n",strerror(errno));
        //     continue;
        // }
        if(ERR == ret )
        {
            LOG_STDOUT(ERROR "MQ_RECV:%s\n",strerror(errno));
            continue;
        }
        switch(queueData.msgID)
        {
            case(LT_MSG_TASK_EXIT):
                continue_flag = 0;
```

```
                LT_LOG(fp,INFO "Logger Task Exit request
from:%s\n",getTaskIdentfierString(queueData.sourceID));
                LOG_STDOUT(INFO "Logger Task Exit request
from:%s\n",getTaskIdentfierString(queueData.sourceID));
                break;

            case(LT_MSG_COMM_MSG):
                LT_LOG(fp,INFO "[%s]
Sender:%s\tCOMM_MSG",queueData.timestamp,getTaskIdentfierString(queueData
.sourceID));
                LT_LOG_COMM(fp,queueData.msgData.commMsg);
                break;

            case(LT_MSG_LOG):
                if(g_loglevel >= queueData.loglevel)
                {
                    #ifdef STDOUT_LOG
                    LOG_STDOUT(INFO "[%s]
Sender:%s\tMsg:%s",queueData.timestamp,getTaskIdentfierString(queueData.s
ourceID),queueData.msgData.msgData);
                    #endif
                    LT_LOG(fp,INFO "[%s]
Sender:%s\tMsg:%s",queueData.timestamp,getTaskIdentfierString(queueData.s
ourceID),queueData.msgData.msgData);
                }
                break;
            case(LT_MSG_TASK_STATUS):
                if(MAIN_TASK_ID == queueData.sourceID)
                {
                    /* Send back task alive response to main task */
                    LT_LOG(fp,INFO "[%s]
Sender:%s\tMsg:%s",queueData.timestamp,getTaskIdentfierString(queueData.s
ourceID),queueData.msgData.msgData);
                    POST_MESSAGE_MAINTASK(&maintaskRsp, "Logger Alive");
                }
                break;

            default:
            LOG_STDOUT(INFO "INVALID QUEUE LOG ID\n");
                break;
        }
    }

}

void* logger_task_callback(void *threadparam)
{
    LOG_STDOUT(INFO "LOGGER TASK STARTED\n");

    char *filename = configdata_getLogpath();
    FILE *fp;
    if(filename)
    {
        fp = logger_task_file_init(filename);
    }
    else
    {
        LOG_STDOUT(WARNING "No filename found from config file\n");
```

```
        fp = logger_task_file_init("project1.log");
    }

    if(NULL == fp)
    {
        LOG_STDOUT(ERROR "LOGGER TASK LOG FILE INIT FAIL\n");
        exit(ERR);
    }

    int ret = logger_task_queue_init();
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "LOGGER TASK INIT%s\n",strerror(errno));
        exit(ERR);
    }

    LOG_STDOUT(INFO "LOGGER TASK INIT COMPLETED\n");
    LT_LOG(fp,INFO "LOGGER TASK INIT COMPLETED\n");
    pthread_barrier_wait(&tasks_barrier);

    #ifdef VALUES
    LOG_STDOUT(INFO "LOGGER TASK UP and RUNNING\n");
    #endif
    #ifdef LOGVALUES
    LT_LOG(fp,INFO "LOGGER TASK UP and RUNNING\n");
    #endif
    /* Process Log queue msg which executes untill the log_task_end flag
is set to true*/
    logger_task_processMsg(fp);

    mq_close(loggertask_q);
    fflush(fp);
    fclose(fp);
    LOG_STDOUT(INFO "Logger Task Exit.\n");
    return (void*)SUCCESS;
}/**
 * @brief
 *
 * @file readConfiguration.c
 * @author Gunj Manseta
 * @date 2018-03-17
 */


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

#include "readConfiguration.h"

#define CONFIG_DATA_NUM 3

#define CONFIG_FILE "configuration.dat"

typedef enum
{
    LOG_PATH_STRING,
```

```
    TASK_SETUP_TIME_SEC_UINT8,
    TASK_ALIVE_TIMEOUT_SEC_UINT8
}CONFIG_DATA_INDEX;

static void* configurationData[CONFIG_DATA_NUM] = {0};

char* configdata_getLogpath()
{
    return ((char*)configurationData[LOG_PATH_STRING]);
}

uint32_t configdata_getSetupTime()
{
    return (*((uint32_t*)configurationData[TASK_SETUP_TIME_SEC_UINT8]));
}
uint32_t configdata_getAliveTimeout()
{
    return
(*((uint32_t*)configurationData[TASK_ALIVE_TIMEOUT_SEC_UINT8]));
}

int configdata_setup()
{
    FILE *fp;
    fp = fopen(CONFIG_FILE, "r");
    if(NULL ==fp)
        return -1;

    configurationData[LOG_PATH_STRING] = (char*)malloc(sizeof(char)*20);
    configurationData[TASK_SETUP_TIME_SEC_UINT8] =
(uint32_t*)malloc(sizeof(uint32_t));
    configurationData[TASK_ALIVE_TIMEOUT_SEC_UINT8] =
(uint32_t*)malloc(sizeof(uint32_t));

    size_t readBytes = fscanf(fp,"%s %u
%u",(char*)configurationData[LOG_PATH_STRING],(uint32_t*)configurationDat
a[TASK_SETUP_TIME_SEC_UINT8],
(uint32_t*)configurationData[TASK_ALIVE_TIMEOUT_SEC_UINT8]);

    #ifdef SELF_TEST
    printf("PATH: %s\n",(char*)configurationData[LOG_PATH_STRING]);
    printf("SETUP:
%u\n",*(uint32_t*)configurationData[TASK_SETUP_TIME_SEC_UINT8]);
    printf("TO:
%u\n",*(uint32_t*)configurationData[TASK_ALIVE_TIMEOUT_SEC_UINT8]);
    #endif

    return 0;
}

void configdata_flush()
{
    for(int i = 0; i <CONFIG_DATA_NUM ; i ++)
    {
        free(configurationData[i]);
        configurationData[i] = NULL;
    }
}
```

```
#ifdef SELF_TEST
int main()
{
    int ret = configdata_setup();
    if(ret) return ret;

    printf("From func: %s\n",configdata_getLogpath());
    printf("From func: %u\n",configdata_getSetupTime());
    printf("From func: %u\n",configdata_getAliveTimeout());

    configdata_flush();
    return 0;
}
#endif/**
 * @brief
 *
 * @file common_helper.c
 * @author Gunj Manseta
 * @date 2018-03-10
 */

#include "common_helper.h"
#include "main_task.h"
#include "logger_task.h"
#include "light_sensor_task.h"
#include "temperature_sensor_task.h"
#include "posixTimer.h"


const char* const task_identifier_string[NUM_CHILD_THREADS+1] =
{
    (const char*)"Logger Task",
    (const char*)"Temperature Task",
    (const char*)"Socket Task",
    (const char*)"Light Task",
    (const char*)"COMM Receiver Task",
    (const char*)"COMM Sender Task",
    (const char*)"Dispatcher Task",
    (const char*)"Main Task",

};

mqd_t get_queue_handle(TASK_IDENTIFIER_T taskid)
{
    mqd_t queueHandle;
    switch(taskid)
    {
        case(MAIN_TASK_ID):
            queueHandle = getHandle_MainTaskQueue();
            break;
        case(LOGGER_TASK_ID):
            queueHandle = getHandle_LoggerTaskQueue();
            break;
        case(LIGHT_TASK_ID):
            queueHandle = getHandle_LightTaskQueue();
            break;
        case(TEMPERATURE_TASK_ID):
```

```
            queueHandle = getHandle_TemperatureTaskQueue();
            break;
        // case(SOCKET_TASK_ID):
        //     queueHandle = getHandle_SocketTaskQueue();
        //     break;
        default:
            queueHandle = 0;
            break;
    }

    return queueHandle;

}

int register_and_start_timer(timer_t *timer_id, uint32_t usec, uint8_t
oneshot, void (*timer_handler)(union sigval), void *handlerArgs)
{
    if(register_timer(timer_id, timer_handler, timer_id) == -1)
      {
          LOG_STDOUT("[ERROR] Register Timer\n");
            return ERR;
      }
      // else
      //    LOG_STDOUT("[INFO] Timer created\n");

      if(start_timer(*timer_id , usec, oneshot) == -1)
      {
            LOG_STDOUT("[ERROR] Start Timer\n");
            return ERR;
      }
      // else
      //    LOG_STDOUT("[INFO] Timer started\n");

}/**
 * @brief
 *
 * @file light_sensor_task.c
 * @author Gunj Manseta
 * @date 2018-03-11
 */

#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>

#include "main_task.h"
#include "logger_task.h"
#include "light_sensor_task.h"
#include "error_data.h"
#include "apds9301_sensor.h"
#include "my_i2c.h"
#include "common_helper.h"


#define MQ_LIGHTTASK_NAME "/lighttask_queue"
```

```
#define LUX_THRESHOLD (50)

static mqd_t lighttask_q;

pthread_mutex_t stateChangeLock;

volatile static DAY_STATE_T isDay;


DAY_STATE_T getLightTask_state()
{
    DAY_STATE_T state;
    pthread_mutex_lock(&stateChangeLock);
    state = isDay;
    pthread_mutex_unlock(&stateChangeLock);
    return state;
}

float getLightTask_lux()
{
    float lux = APDS9301_getLux();
    return lux;
}

static void timer_handler_getAndUpdateState(union sigval sig)
{
    DAY_STATE_T state;

    DEFINE_LOG_STRUCT(logtaskstruct,LT_MSG_LOG,LIGHT_TASK_ID);

    float lux = APDS9301_getLux();
    if(lux < 0)
    {
        LOG_STDOUT(ERROR "Light sensor inactive\n");
        POST_MESSAGE_LOGTASK(&logtaskstruct,ERROR "Light sensor
inactive\n");
        return;
    }
    else
    {
        #ifdef VALUES
        LOG_STDOUT(INFO "Lux: %.03f\n",lux);
        #endif

        #ifdef LOGVALUES
        POST_MESSAGE_LOGTASK(&logtaskstruct,INFO "Lux: %.03f\n",lux);
        #endif
    }

    (lux < LUX_THRESHOLD) ? (state = NIGHT) : (state = DAY);
    #ifdef VALUES
    LOG_STDOUT(INFO "State: %s\n", ((state == DAY)?"DAY":"NIGHT"));
    #endif

    pthread_mutex_lock(&stateChangeLock);
    isDay = state;
    pthread_mutex_unlock(&stateChangeLock);
```

```
}

mqd_t getHandle_LightTaskQueue()
{
    return lighttask_q;
}

/**
 * @brief
 *
 * @return int
 */
int light_task_queue_init()
{
    struct mq_attr lighttaskQ_attr = {
        .mq_msgsize = sizeof(LIGHTTASKQ_MSG_T),
        .mq_maxmsg = 128,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_LIGHTTASK_NAME);
    lighttask_q = mq_open(MQ_LIGHTTASK_NAME, O_CREAT | O_RDWR, 0666,
&lighttaskQ_attr);

    return lighttask_q;;
}

void light_task_processMsg()
{
    int ret,prio;
    LIGHTTASKQ_MSG_T queueData = {0};
    DEFINE_MAINTASK_STRUCT(maintaskRsp,MT_MSG_STATUS_RSP,LIGHT_TASK_ID);
    DEFINE_LOG_STRUCT(logtaskstruct,LT_MSG_LOG,LIGHT_TASK_ID);
    //struct timespec recv_timeout = {0};
    uint8_t continue_flag = 1;
    /* Uncomment to check the keep alive feature. Only a cancellable
function defined by POSIX can be used below as the we are using
pthread_cancel */
    //sleep(10);
    while(continue_flag)
    {
        memset(&queueData,0,sizeof(queueData));
        //clock_gettime(CLOCK_REALTIME, &recv_timeout);
        //recv_timeout.tv_sec += 3;
        //ret =
mq_timedreceive(lighttask_q,(char*)&(queueData),sizeof(queueData),&prio,
&recv_timeout);
        ret =
mq_receive(lighttask_q,(char*)&(queueData),sizeof(queueData),&prio);
        if(ERR == ret)
        {
            LOG_STDOUT(ERROR "MQ_RECV:%s\n",strerror(errno));
            continue;
        }
        switch(queueData.msgID)
        {
            case(LIGHT_MSG_TASK_STATUS):
```

```
                /* Send back task alive response to main task */
                POST_MESSAGE_LOGTASK(&logtaskstruct,INFO "ALIVE STATUS
by:%s\n",getTaskIdentfierString(queueData.sourceID));
                POST_MESSAGE_MAINTASK(&maintaskRsp, "Light sensor task
Alive");
                break;
            case(LIGHT_MSG_TASK_GET_STATE):

                // (queueData.packet.reg_value != NULL)?
(*queueData.packet.reg_value = getLightTask_state()) : 0;
                // queueData.packet.buffLen = 1;
                // (queueData.packet.is_sync) ?
(sem_post(queueData.packet.sync_semaphore)): 0;

                break;
            case(LIGHT_MSG_TASK_READ_DATA):
                break;
            case(LIGHT_MSG_TASK_WRITE_CMD):
                break;
            case(LIGHT_MSG_TASK_POWERDOWN):
                APDS9301_powerdown();
                break;
            case(LIGHT_MSG_TASK_POWERUP):
                APDS9301_poweron();
                break;
            case(LIGHT_MSG_TASK_EXIT):
                continue_flag = 0;
                LOG_STDOUT(INFO "Light Task Exit request
from:%s\n",getTaskIdentfierString(queueData.sourceID));
                POST_MESSAGE_LOGTASK(&logtaskstruct,INFO "Light Task Exit
request from:%s\n",getTaskIdentfierString(queueData.sourceID));
                break;
            default:
                break;
        }
    }

}

int light_task_sensorUP(I2C_MASTER_HANDLE_T *i2c)
{
    int ret = 0;
    ret = I2Cmaster_Init(i2c);
    if(ret !=0)
    {
        printErrorCode(ret);
        LOG_STDOUT(ERROR "I2C Master init failed\n");
    }

    ret = APDS9301_poweron();
    if(ret == 0) LOG_STDOUT(INFO "[OK] Sensor powered ON\n");

    ret = APDS9301_test();
    if(ret == 0) {LOG_STDOUT(INFO "[OK] Sensor Test\n");}
    else {LOG_STDOUT(INFO "[FAIL] Sensor Test\n");}

    return ret;
}
```

```
int light_task_sensorDOWN(I2C_MASTER_HANDLE_T *i2c)
{
    int ret = 0;
    ret = APDS9301_powerdown();
    if(ret == 0) LOG_STDOUT(INFO "Sensor powered DOWN\n");
    ret = I2Cmaster_Destroy(i2c);
    if(ret != 0)
    {
        printErrorCode(ret);
        LOG_STDOUT(WARNING "I2C Master destroy failed\n");
    }

    return ret;
}

void* light_task_callback(void *threadparam)
{
    LOG_STDOUT(INFO "LIGHT TASK STARTED\n");

    int ret = light_task_queue_init();
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "LIGHT TASK QUEUE INIT:%s\n",strerror(errno));
        exit(ERR);
    }

    I2C_MASTER_HANDLE_T i2c;
    ret = light_task_sensorUP(&i2c);
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "LIGHT TASK SENSOR INIT:%s\n",strerror(errno));
        goto FAIL_EXIT_SENSOR;
    }


    LOG_STDOUT(INFO "[OK] LIGHT TASK INIT COMPLETED\n");
    pthread_barrier_wait(&tasks_barrier);

    /* Registering a timer for 2 sec to update the state of the snesor
value by getting the lux value from the sensor*/
    timer_t timer_id;
    if(ERR == register_and_start_timer(&timer_id, 2*MICROSEC, 0,
timer_handler_getAndUpdateState, &timer_id))
    {
        // LOG_STDOUT(ERROR "Timer Error\n");
        goto FAIL_EXIT;
    }

    /* Process Log queue msg which executes untill the log_task_end flag
is set to true*/
    light_task_processMsg();

    ret = delete_timer(timer_id);
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "LIGHT TASK DELETE TIMER:%s\n",strerror(errno));
    }
```

```
FAIL_EXIT:

    light_task_sensorDOWN(&i2c);

FAIL_EXIT_SENSOR:
    mq_close(lighttask_q);
    LOG_STDOUT(INFO "Light task exit.\n");
    return SUCCESS;
}/**
 * @brief
 *
 * @file dispatcher_task.c
 * @author Gunj Manseta
 * @date 2018-04-26
 */
#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>
#include <signal.h>

#include "main_task.h"
#include "common_helper.h"
#include "logger_task.h"
#include "dispatcher_task.h"
#include "communication_object.h"
#include "communication_interface.h"

#define MQ_DISPATCHER_TASK_NAME "/dispatcher_task_queue"


static mqd_t dispatcher_task_q;

mqd_t getHandle_DispatcherTaskQueue()
{
    return dispatcher_task_q;
}

int dispatcher_task_queue_init()
{
    struct mq_attr dispatcher_taskQ_attr = {
        .mq_msgsize = sizeof(COMM_MSG_T),
        .mq_maxmsg = 128,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_DISPATCHER_TASK_NAME);
    dispatcher_task_q = mq_open(MQ_DISPATCHER_TASK_NAME, O_CREAT |
O_RDWR, 0666, &dispatcher_taskQ_attr);

    return dispatcher_task_q;
}

/* from teh socket task */
```

```
extern uint8_t gotDistance;
extern COMM_MSG_T socket_comm_msg;

/* Waits on the queue items containing the comm mgs, process it depending
on the msg id and dst id */
/* Call function accordingly */
void dispatcher_task_processMsg()
{
    int ret,prio;
    COMM_MSG_T queueData = {0};

DEFINE_MAINTASK_STRUCT(maintaskRsp,MT_MSG_STATUS_RSP,DISPATCHER_TASK_ID);
    DEFINE_LOG_STRUCT(log_struct,LT_MSG_COMM_MSG,DISPATCHER_TASK_ID);
    //struct timespec recv_timeout = {0};
    uint8_t continue_flag= 1;
    while(continue_flag)
    {
        memset(&queueData,0,sizeof(queueData));
        // clock_gettime(CLOCK_REALTIME, &recv_timeout);
        // recv_timeout.tv_sec += 3;
        // ret =
mq_timedreceive(loggertask_q,(char*)&(queueData),sizeof(queueData),&prio,
&recv_timeout);
        ret =
mq_receive(dispatcher_task_q,(char*)&(queueData),sizeof(queueData),&prio)
;
        // if(ERR == ret && ETIMEDOUT == errno)
        // {
        //     //LOG_STDOUT(ERROR "MQ_RECV
TIMEOUT:%s\n",strerror(errno));
        //     continue;
        // }
        if(ERR == ret )
        {
            LOG_STDOUT(ERROR "MQ_RECV:%s\n",strerror(errno));
            continue;
        }
        if((0xFF) == queueData.msg_id)
        {
            if(queueData.dst_brd_id == BBG_BOARD_ID)
            {
                continue_flag = 0;
                continue;
            }
        }

        if(!verifyCheckSum(&queueData))
        {
            LOG_STDOUT(INFO "COMM_MSG Checksum failed\n");
            continue;
        }

        switch(queueData.dst_id)
        {
            case BBG_LOGGER_MODULE:
                POST_COMM_MSG_LOGTASK(&log_struct,queueData);
                break;
            case BBG_COMM_MODULE:
```

```
                break;
            case BBG_SOCKET_MODULE:

memcpy(&socket_comm_msg,&queueData,sizeof(socket_comm_msg));
                gotDistance = 1;
                break;
            default:
                LOG_STDOUT(INFO "Invalid msg id\n");
                break;

        }
    }
    gotDistance = 1;
}

/* Create the entry function */
void* dispatcher_task_callback(void *threadparam)
{
    LOG_STDOUT(INFO "DISPATCHER STARTED\n");
    int ret = dispatcher_task_queue_init();
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "DISPATCHER INIT%s\n",strerror(errno));
        //exit(ERR);
        goto EXIT;
    }

    LOG_STDOUT(INFO "DISPATCHER INIT DONE\n");
    pthread_barrier_wait(&tasks_barrier);

    dispatcher_task_processMsg();

    mq_close(dispatcher_task_q);

EXIT:
    LOG_STDOUT(INFO "DISPATCHER TASK EXIT\n");
    return SUCCESS;
}
/**
 * @brief
 *
 * @file posixTimer.c
 * @author Gunj Manseta
 * @date 2018-03-18
 */

#include "posixTimer.h"

#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <signal.h>
#include <time.h>
#include <sys/types.h>
#include <string.h>

int register_timer(timer_t *timer_id, void (*timer_handler)(union
sigval), void *handlerArgs)
```

```
{

	if(NULL == timer_id)
		return -1;

	struct sigevent sige;

	/*SIGEV_THREAD will call the handler as if it was a new thread */
	sige.sigev_notify = SIGEV_THREAD;
	sige.sigev_notify_function = timer_handler;
//	sige.sigev_value.sival_ptr = timer_id;
	sige.sigev_value.sival_ptr = handlerArgs;
	sige.sigev_notify_attributes = NULL;

	int ret = timer_create(CLOCK_REALTIME, &sige, timer_id);

	return ret;
}

int start_timer(timer_t timer_id , uint64_t time_usec, uint8_t oneshot)
{
	// if(NULL == timer_id)
	//	return -1;

	struct itimerspec ts;

	ts.it_value.tv_sec = time_usec / MICROSEC;
	ts.it_value.tv_nsec = (time_usec % MICROSEC) * 1000;
	if(1 == oneshot)
	{
		ts.it_interval.tv_sec = 0;
		ts.it_interval.tv_nsec = 0;
	}
	else
	{
		ts.it_interval.tv_sec = ts.it_value.tv_sec;
		ts.it_interval.tv_nsec = ts.it_value.tv_nsec;
	}

	int ret = timer_settime(timer_id, 0, &ts, 0);

	return ret;
}

int stop_timer(timer_t timer_id)
{
	// if(NULL == timer_id)
	//	return -1;

	struct itimerspec ts;

	ts.it_value.tv_sec = 0;
	ts.it_value.tv_nsec = 0;
	ts.it_interval.tv_sec = 0;
	ts.it_interval.tv_nsec = 0;

	int ret = timer_settime(timer_id, 0, &ts, 0);
```

```
      return ret;
}


int delete_timer(timer_t timer_id)
{
      // if(NULL == timer_id)
      //    return -1;

      int ret = timer_delete(timer_id);

      return ret;


}
/**
 * @brief Credits
:https://github.com/adarqui/darqbot/blob/master/test/how-to-generate-a-
stacktrace-when-my-gcc-c-app-crashes
 *
 * @file seg_fault_signal.c
 * @author https://github.com/adarqui/darqbot/blob/master/test/how-to-
generate-a-stacktrace-when-my-gcc-c-app-crashes
 * @date 2018-04-27
 */

#include <execinfo.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ucontext.h>
#include <stdio.h>
#include <unistd.h>


/* This structure mirrors the one found in /usr/include/asm/ucontext.h */
typedef struct _sig_ucontext {
unsigned long     uc_flags;
struct ucontext   *uc_link;
stack_t           uc_stack;
struct sigcontext uc_mcontext;
sigset_t          uc_sigmask;
} sig_ucontext_t;

void crit_err_hdlr(int sig_num, siginfo_t * info, void * ucontext)
{
    void *             array[50];
    void *             caller_address = 0;
    char **            messages;
    int                size, i;
    sig_ucontext_t *   uc;

    uc = (sig_ucontext_t *)ucontext;

    /* Get the address at the time the signal was raised from the EIP
(x86) */
    caller_address = (void *) uc->uc_mcontext.arm_ip;
```

```
    fprintf(stderr, "signal %d (%s), address is %p from %p\n",
        sig_num, strsignal(sig_num), info->si_addr,
        (void *)caller_address);

    size = backtrace(array, 50);

    /* overwrite sigaction with caller's address */
    array[1] = caller_address;

    messages = backtrace_symbols(array, size);

    /* skip first stack frame (points here) */
    for (i = 1; i < size && messages != NULL; ++i)
    {
        fprintf(stderr, "[bt]: (%d) %s\n", i, messages[i]);
    }

    free(messages);

    exit(EXIT_FAILURE);
}


void handler(int sig)
{
    void *array[50];
    size_t size;

    // get void*'s for all entries on the stack
    size = backtrace(array, 50);
    fprintf(stderr, "Error: size %u:\n", size);

    // print out all the frames to stderr
    fprintf(stderr, "Error: signal %d:\n", sig);
    backtrace_symbols_fd(array, size, STDERR_FILENO);
    exit(EXIT_FAILURE);
}


void install_segfault_signal()
{
     struct sigaction sigact;

    sigact.sa_sigaction = crit_err_hdlr;
    sigact.sa_flags = SA_RESTART | SA_SIGINFO;

    if (sigaction(SIGSEGV, &sigact, (struct sigaction *)NULL) != 0)
    {
        fprintf(stderr, "error setting signal handler for %d
(%s)\n",SIGSEGV, strsignal(SIGSEGV));
        exit(EXIT_FAILURE);
    }

    //signal(SIGSEGV, handler);
}/*
 * spi.c
 *
```

```
 *  Created on: Dec 1, 2017
 *      Author: Gunj Manseta
 */

#if 1
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>

#include "spi.h"


#define CHECK_SPI_NUM(spi)       ({if(spi > NUM_SPI_BUS-1)   \
                                  return 1;})

static uint32_t opened[NUM_SPI_BUS] = {0};

SPI_Type SPI[NUM_SPI_BUS] = {NULL,NULL,NULL,NULL};

static SPI_t SPI_release(SPI_t spi)
{
    if(spi > NUM_SPI_BUS-1)
        return -1;

    mraa_result_t status = mraa_spi_stop(SPI[spi]);
    if (status != MRAA_SUCCESS)
        return -1;
    return spi;
}

SPI_t SPI_init(SPI_t spi)
{
    if(spi > NUM_SPI_BUS-1)
        return -1;

    if(opened[spi] && SPI[spi] != NULL)
    {
        opened[spi]++;
        return spi;
    }
    mraa_result_t status = MRAA_SUCCESS;

    mraa_spi_context spi_context = mraa_spi_init(spi);
    if(spi_context == NULL)
    {
        return 0;
    }
    SPI[spi] = spi_context;
    status = mraa_spi_frequency(SPI[spi], SPI_2MZ);
    if (status != MRAA_SUCCESS)
    {
        return SPI_release(spi);
    }

    opened[spi]++;
    return spi;
}
```

```
void SPI_GPIO_init(SPI_t spi)
{
}

SPI_t SPI_disable(SPI_t spi)
{
    if(spi > NUM_SPI_BUS-1)
        return -1;

    opened[spi]--;
    if(opened[spi])
    {
        return spi;
    }
    else
    {
        return SPI_release(spi);
    }
}

int8_t SPI_write_packet(SPI_t spi, uint8_t* p, size_t length)
{
    CHECK_SPI_NUM(spi);

      uint8_t i=0;
      while (i<length)
      {
            SPI_write_byte(spi, *(p+i));
            ++i;
      }
    return length;
}

int32_t  SPI_read_packet(SPI_t spi, uint8_t* p, size_t length)
{
    CHECK_SPI_NUM(spi);
      uint8_t i=0;
      while (i<length)
      {
            *(p+i) = SPI_read_byte(spi);
            ++i;
      }
    return length;
}

void SPI0_IRQHandler()
{

}

#endif/**
 * @brief
 *
 * @file comm_recv_task.c
 * @author Gunj Manseta
 * @date 2018-04-26
 */
```

```
#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>
#include <signal.h>

#include "error_data.h"
#include "common_helper.h"
#include "comm_recv_task.h"
#include "communication_object.h"
#include "communication_interface.h"
#include "dispatcher_task.h"

volatile sig_atomic_t comm_recv_task_exit = 0;

static int8_t getFrame();

/* Create the entry function */
void* comm_recv_task_callback(void *threadparam)
{
    UART_FD_T fd = COMM_INIT();
    int32_t retrycount = 0;
    int32_t checksum_incorrect = 0;
    if(fd < 0)
    {
        /* LOG ERROR */
        return (void*)ERROR;
    }

    LOG_STDOUT(INFO "COMM RECV TASK INIT COMPLETED\n");

    pthread_barrier_wait(&tasks_barrier);

    COMM_MSG_T recv_comm_msg = {0};
    while(!comm_recv_task_exit)
    {
        memset(&recv_comm_msg, 0 , sizeof(recv_comm_msg));
        int32_t ret = comm_recvUART(&recv_comm_msg);
        /* Some error */
        //printf("RET:%d Retry:%d\n",ret,retrycount);
        if(ret == -1)
        {
            /* LOG error */
            LOG_STDOUT(ERROR "COMM RECV\n");
        }
        else if(ret > 0)
        {
            /* Send to dispatcher */
            uint16_t check = getCheckSum(&recv_comm_msg);
            if(check != recv_comm_msg.checksum)
            {
                checksum_incorrect++;
                if(checksum_incorrect > 3)
                {
                    UART_flush();
                    checksum_incorrect = 0;
```

```
                }
                continue;
            }
            retrycount = 0;
            if(recv_comm_msg.dst_brd_id != BBG_BOARD_ID)
            {
                LOG_STDOUT(INFO "Not my Board ID. I am not touching
it.\n");
                continue;
            }
            if(recv_comm_msg.msg_id == MSG_ID_OBJECT_DETECTED)
            {
                if(getFrame())
                {
                    LOG_STDOUT(ERROR "Frame save error.\n");
                }
                else
                {
                    LOG_STDOUT(INFO "Frame saved successfully.\n");
                }

            }
            POST_MESSAGE_DISPATCHERTASK(&recv_comm_msg);
            LOG_STDOUT_COMM(recv_comm_msg);
            /* LOG_STDOUT(INFO "\n*******\n\
            SRCID:%u, SRC_BRDID:%u, DST_ID:%u, DST_BRDID:%u MSGID:%u\n\
            SensorVal: %f MSG:%s\n\
            Checksum:%u ?= %u\n********\n",\
            recv_comm_msg.src_id, recv_comm_msg.src_brd_id,
recv_comm_msg.dst_id,recv_comm_msg.dst_brd_id,recv_comm_msg.msg_id,recv_c
omm_msg.data.distance_cm,recv_comm_msg.message,recv_comm_msg.checksum,
check );
            */
        }
        else
        {
            retrycount++;
            if(retrycount > 100)
            {
                LOG_STDOUT(WARNING "TIVA CONNECTED????\n");
                retrycount = 0;
            }
        }
    }

    COMM_DEINIT(fd);

    LOG_STDOUT(INFO "COMM RECV Task Exit\n");
    return (void*)SUCCESS;

}

#define p320    4000
#define p640    9000

static int8_t getFrame()
{
    /* uint8_t buffer_320p[3600] = {0};
```

```
    uint8_t buffer_640p[8500] = {0}; */
    uint8_t *buffer = (uint8_t*)malloc(sizeof(uint8_t)*p640);
    //uint8_t *buffer = buffer_640p;
    uint8_t temp = 0, temp_last = 0;
    uint32_t len = 0;
    int i = 0;
    uint8_t done = 0;
    uint32_t retry = 0;
    uint8_t getpix = 0;
    uint8_t header = 0;
    //static uint32_t image_count = 0;
    while(1)
    {
        int32_t ret = UART_read((uint8_t*)&getpix,1);
        /* Some error */
        //printf("RET:%d Retry:%d\n",ret,retrycount);
        if(ret == -1)
        {
            /* LOG error */
            LOG_STDOUT(ERROR "Frame Recv\n");
        }
        else if(ret > 0 && (getpix == 0xFF || header == 1) )
        {
            if(ret == 1)
            {
                if(getpix == 0xFF)
                    header = 1;
                buffer[i] = getpix;
                //printf("0x%x ",buffer[i]);
                if(temp_last == 0xFF && buffer[i] == 0xD9)
                {
                    LOG_STDOUT(INFO "EOF found\n");
                    done = 1;
                    break;
                }
                temp_last  = buffer[i];
                i++;
            }
        }
        else
        {
            retry++;
            if(retry > 1024)
            {
                //printf("Connected?\n");
                break;
            }
        }
    }

    if(done)
    {
        char newFilename[25] = {0};

snprintf(newFilename,sizeof(newFilename),"%s_%u.%s","image",((unsigned)ti
me(NULL)&0xFFFFFF),"jpg");
        FILE *fp = fopen(newFilename, "wb");
        fwrite(buffer,i,1,fp);
```

```
        fclose(fp);
        //image_count++;
        free(buffer);
        return 0;
    }
    free(buffer);
    return 1;
}

/**
 * @brief Implementation file for the driver functions of the NRF240L
 *
 * @file nordic_driver.c
 * @author Gunj Manseta
 * @date 2018-04-28
 */

#if 1

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

#include "my_uart.h"

#include "nordic_driver.h"
#include "mraa/spi.h"
#include "mraa/gpio.h"

//Commands Byte
#define NORDIC_TXFIFO_FLUSH_CMD   (0xE1)
#define NORDIC_RXFIFO_FLUSH_CMD   (0xE2)
#define NORDIC_W_TXPAYLD_CMD (0xA0)
#define NORDIC_R_RXPAYLD_CMD (0x61)
#define NORDIC_ACTIVATE_CMD       (0x50)
#define NORDIC_ACTIVATE_DATA (0x73)
#define NORDIC_RXPAYLD_W_CMD (0x60)
#define NORDIC_NOP                      (0xFF)
//Register Addresses
#define NORDIC_CONFIG_REG         (0x00)
#define NORDIC_EN_AA_REG        (0x01)
#define NORDIC_EN_RXADDR_REG    (0x02)
#define NODIC_SETUP_RETR_REG    (0x04)
#define NORDIC_STATUS_REG         (0x07)
#define NORDIC_RF_SETUP_REG       (0x06)
#define NORDIC_RF_CH_REG          (0x05)
#define NORDIC_TX_ADDR_REG        (0x10)
#define NORDIC_TX_ADDR_LEN        (5)

#define NORDIC_RX_ADDR_P0_REG   (0x0A)
#define NORDIC_RX_ADDR_P1_REG   (0x0B)
#define NORDIC_RX_ADDR_P2_REG   (0x0C)
#define NORDIC_RX_ADDR_P3_REG   (0x0D)
#define NORDIC_RX_ADDR_P4_REG   (0x0E)
#define NORDIC_RX_ADDR_P5_REG   (0x0F)

#define NORDIC_FIFO_STATUS_REG    (0x17)
#define NORDIC_RX_PW_P0_REG       (0x11)
```

```
#define DEFAULT_TX_ADDRESS_1B      (0xE7)
#define DEFAULT_TX_ADDRESS_2B      (0xE7)
#define DEFAULT_TX_ADDRESS_3B      (0xE7)
#define DEFAULT_TX_ADDRESS_4B      (0xE7)
#define DEFAULT_TX_ADDRESS_5B      (0xE7)

//Masks
#define NORDIC_CONFIG_MAX_RT_MASK       4
#define NORDIC_CONFIG_MAX_RT_INT(x)
((((uint8_t)x)<<NORDIC_CONFIG_MAX_RT_MASK)&(1<<NORDIC_CONFIG_MAX_RT_MASK)
)

#define NORDIC_CONFIG_RX_DR_MASK        6
#define NORDIC_CONFIG_RX_DR_INT(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_RX_DR_MASK)&(1<<NORDIC_CONFIG_RX_DR_M
ASK))

#define NORDIC_CONFIG_TX_DS_MASK        5
#define NORDIC_CONFIG_TX_DS_INT(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_TX_DS_MASK)&(1<<NORDIC_CONFIG_TX_DS_M
ASK))

#define NORDIC_CONFIG_PWR_UP_MASK       1
#define NORDIC_CONFIG_PWR_UP(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_PWR_UP_MASK)&(1<<NORDIC_CONFIG_PWR_UP
_MASK))

#define NORDIC_CONFIG_PRIM_RX_MASK      0
#define NORDIC_CONFIG_PRIM_RX(x)
      ((((uint8_t)x)<<NORDIC_CONFIG_PRIM_RX_MASK)&(1<<NORDIC_CONFIG_PRIM_
RX_MASK))


#define NORDIC_STATUS_TX_FULL_MASK            (1<<0)
#define NORDIC_FIFO_STATUS_TX_FULL_MASK       (1<<5)
#define NORDIC_FIFO_STATUS_RX_FULL_MASK       (1<<1)
#define NORDIC_FIFO_STATUS_TX_EMPTY_MASK      (1<<4)
#define NORDIC_FIFO_STATUS_RX_EMPTY_MASK      (0<<5)

#define NORDIC_INT_MAXRT_MASK   (1<<3)
#define NORDIC_INT_TXDS_MASK    (1<<4)
#define NORDIC_INT_TXDR_MASK    (1<<5)

volatile uint8_t txconfigured = 0;
volatile uint8_t rxconfigured = 0;

volatile uint8_t transmitted = 0;
volatile uint8_t received = 0;
volatile uint8_t retry_error = 0;

static uint8_t using_interrupt = 0;



#define NRF_SPI_BUS   0

void NRF_IntHandler(void *args);
```

```
static NRF_INT_HANDLER_T user_handler;

#define NORDIC_CE_PIN_MRAA      73
#define NORDIC_CSN_PIN_MRAA     70
#define NORDIC_IRQ_PIN_MRAA     69

mraa_gpio_context NRF_CSN_GPIO = 0;
mraa_gpio_context NRF_CE_GPIO = 0;
static mraa_gpio_context NRF_IRQ_GPIO = 0;


int8_t NRF_gpioInit()
{
    mraa_result_t status = MRAA_SUCCESS;
    NRF_CSN_GPIO = mraa_gpio_init(NORDIC_CSN_PIN_MRAA);
    if(NRF_CSN_GPIO == NULL)
    {
        goto ERR_CSN;
    }

    NRF_CE_GPIO = mraa_gpio_init(NORDIC_CE_PIN_MRAA);
    if(NRF_CE_GPIO == NULL)
    {
        goto ERR_CE;
    }
    NRF_IRQ_GPIO = mraa_gpio_init(NORDIC_IRQ_PIN_MRAA);
    if(NRF_IRQ_GPIO == NULL)
    {
        goto ERR_IRQ;
    }

    status = mraa_gpio_dir(NRF_CSN_GPIO, MRAA_GPIO_OUT);
    if (status != MRAA_SUCCESS)
    {
        goto ERR;
    }

    status = mraa_gpio_dir(NRF_CE_GPIO, MRAA_GPIO_OUT);
    if (status != MRAA_SUCCESS)
    {
        goto ERR;
    }

    status = mraa_gpio_dir(NRF_IRQ_GPIO, MRAA_GPIO_IN);
    if (status != MRAA_SUCCESS)
    {
        goto ERR;
    }
    return 1;

ERR:
    status = mraa_gpio_close(NRF_IRQ_GPIO);
ERR_IRQ:
    status = mraa_gpio_close(NRF_CE_GPIO);
ERR_CE:
    status = mraa_gpio_close(NRF_CSN_GPIO);
ERR_CSN:
```

```
    return -1;
}

int8_t NRF_moduleInit(uint8_t use_interrupt, NRF_INT_HANDLER_T handler)
{
    SPI_clock_init(NRF_SPI_BUS, 0);
    if(SPI_init(NRF_SPI_BUS))
        return -1;

    DelayMs(1);

    if(NRF_gpioInit())
        return -1;

    if(use_interrupt)
    {
        using_interrupt = 1;
        user_handler = handler;
        mraa_result_t  status = mraa_gpio_isr(NRF_IRQ_GPIO,
MRAA_GPIO_EDGE_FALLING, &NRF_IntHandler, NULL);
        if(status != MRAA_SUCCESS)
        {
            status = mraa_gpio_close(NRF_IRQ_GPIO);
            status = mraa_gpio_close(NRF_CE_GPIO);
            status = mraa_gpio_close(NRF_CSN_GPIO);
            using_interrupt = 0;
            return -1;
        }
    }
    else
    {
        using_interrupt = 0;
    }
    return 1;
}

void NRF_moduleSetup(NRF_DataRate_t DR, NRF_Power_t power)
{
    //Clearing all interrupts
    NRF_write_status(0);
    //Disabling all interrupts and init in power down TX mode
    NRF_write_config(0x78);
    NRF_write_rf_ch(44);
    NRF_write_rf_setup((power<<1) | (DR<<3) | 1);
    //ADDR LEN as 5bytes
    NRF_write_register(0x03, 0x03);
    DelayMs(1);
}

void NRF_moduleDisable()
{
    using_interrupt = 0;
    uint8_t config = NRF_read_config();
    NRF_write_config(config & ~NORDIC_CONFIG_PWR_UP(1));
      SPI_disable(NRF_SPI_BUS);
    mraa_result_t status = mraa_gpio_close(NRF_IRQ_GPIO);
    status = mraa_gpio_close(NRF_CE_GPIO);
    status = mraa_gpio_close(NRF_CSN_GPIO);
```

```
}

uint8_t NRF_read_register(uint8_t regAdd)
{
	//SPI_clear_RXbuffer(NRF_SPI_BUS);//used to clear the previously
value in the RX FIFO
	uint8_t readValue = 0;

	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(NRF_SPI_BUS,regAdd);
	readValue = SPI_read_byte(NRF_SPI_BUS);

	//Marking the end of transaction by CSN high
	NRF_chip_disable();

	return readValue;
}

void NRF_write_command(uint8_t command)
{
	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	SPI_write_byte(NRF_SPI_BUS,command);
	//SPI_read_byte(NRF_SPI_BUS);

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

void NRF_write_register(uint8_t regAdd, uint8_t value)
{
	//SPI_clear_RXbuffer(NRF_SPI_BUS);//used to clear the previously
value in the RX FIFO

	//CSN High to low for new command
	NRF_chip_disable();
	NRF_chip_enable();

	uint8_t ret = SPI_write_byte(NRF_SPI_BUS,regAdd | 0x20);
	//SPI_read_byte(NRF_SPI_BUS);	//used to clear the previously
value in the RX FIFO
	ret = SPI_write_byte(NRF_SPI_BUS,value);
	//SPI_read_byte(NRF_SPI_BUS);	//used to clear the previously
value in the RX FIFO

	//Marking the end of transaction by CSN high
	NRF_chip_disable();
}

void NRF_write_status(uint8_t statusValue)
{
	NRF_write_register(NORDIC_STATUS_REG, statusValue);
}
```

```
uint8_t NRF_read_status()
{
    uint8_t readValue = 0;

    //CSN High to low for new command
    NRF_chip_disable();
    NRF_chip_enable();

    readValue = SPI_write_byte(NRF_SPI_BUS,NORDIC_NOP);
    //readValue = SPI_read_byte(NRF_SPI_BUS);   //used to clear the
previously value in the RX FIFO

    //Marking the end of transaction by CSN high
    NRF_chip_disable();

    return readValue;
}

void NRF_write_config(uint8_t configValue)
{
     NRF_write_register(NORDIC_CONFIG_REG, configValue);
}

uint8_t NRF_read_config()
{
     return NRF_read_register(NORDIC_CONFIG_REG);
}

uint8_t NRF_read_rf_setup()
{
     return NRF_read_register(NORDIC_RF_SETUP_REG);
}

void NRF_write_rf_setup(uint8_t rfSetupValue)
{
     NRF_write_register(NORDIC_RF_SETUP_REG, rfSetupValue);
}

uint8_t NRF_read_rf_ch()
{
     return NRF_read_register(NORDIC_RF_CH_REG);
}

void NRF_write_rf_ch(uint8_t channel)
{
     NRF_write_register(NORDIC_RF_CH_REG, channel);
}

void NRF_write_En_AA(uint8_t data)
{
    NRF_write_register(NORDIC_EN_AA_REG, data);
}

uint8_t NRF_read_En_AA()
{
    return NRF_read_register(NORDIC_EN_AA_REG);
}
```

```
void NRF_write_setup_retry(uint8_t data)
{
    NRF_write_register(NODIC_SETUP_RETR_REG, data);
}

uint8_t NRF_read_setup_retry()
{
    return NRF_read_register(NODIC_SETUP_RETR_REG);
}

void NRF_read_TX_ADDR(uint8_t *address)
{
      uint8_t i = 0;

      NRF_chip_disable();
      NRF_chip_enable();

      SPI_write_byte(NRF_SPI_BUS,NORDIC_TX_ADDR_REG);
      //SPI_read_byte(NRF_SPI_BUS);     //used to clear the previously
value in the RX FIFO
      //SPI_read_byte(NRF_SPI_BUS);     //used to clear the previously
value in the RX FIFO
      while(i < NORDIC_TX_ADDR_LEN)
      {
            SPI_write_byte(NRF_SPI_BUS, 0xFF);//Dummy to get the data
            *(address+i) = SPI_read_byte(NRF_SPI_BUS);
            i++;
      }

      NRF_chip_disable();
}

void NRF_write_TX_ADDR(uint8_t * tx_addr)
{
      NRF_chip_disable();
      NRF_chip_enable();

      SPI_write_byte(NRF_SPI_BUS,NORDIC_TX_ADDR_REG | 0x20);
      //SPI_read_byte(NRF_SPI_BUS);     //used to clear the previously
value in the RX FIFO
      SPI_write_packet(NRF_SPI_BUS,tx_addr,NORDIC_TX_ADDR_LEN);
      //SPI_flushRXFIFO(NRF_SPI_BUS);

      NRF_chip_disable();
}

void NRF_read_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *address)
{
    if(pipe_num > 5)
        return;
//    uint8_t i = 0;

    NRF_chip_disable();
    NRF_chip_enable();

    SPI_write_byte(NRF_SPI_BUS,(NORDIC_RX_ADDR_P0_REG + pipe_num));
    size_t ADDR_LEN = NORDIC_TX_ADDR_LEN;
```

```
    pipe_num > 2 ? ADDR_LEN = 1: 0;
    SPI_read_packet(NRF_SPI_BUS, address, ADDR_LEN);

    NRF_chip_disable();
}

void NRF_write_RX_PIPE_ADDR(uint8_t pipe_num, uint8_t *rx_addr)
{
    if(pipe_num > 5)
        return;

    NRF_chip_disable();
    NRF_chip_enable();

    SPI_write_byte(NRF_SPI_BUS,(NORDIC_RX_ADDR_P0_REG + pipe_num) |
0x20);
    //SPI_read_byte(NRF_SPI_BUS);   //used to clear the previously value
in the RX FIFO
    size_t ADDR_LEN = NORDIC_TX_ADDR_LEN;
    pipe_num > 1 ? ADDR_LEN = 1: 0;
    SPI_write_packet(NRF_SPI_BUS,rx_addr,ADDR_LEN);
    //SPI_flushRXFIFO(NRF_SPI_BUS);

    NRF_chip_disable();
}


uint8_t NRF_read_fifo_status()
{
     return NRF_read_register(NORDIC_FIFO_STATUS_REG);
}

void NRF_flush_tx_fifo()
{
     NRF_write_command(NORDIC_TXFIFO_FLUSH_CMD);
}

void NRF_flush_rx_fifo()
{
     NRF_write_command(NORDIC_RXFIFO_FLUSH_CMD);
}

void NRF_activate_cmd()
{
     NRF_write_register(NORDIC_ACTIVATE_CMD, NORDIC_ACTIVATE_DATA);
}

void NRF_enable_RX_PIPE(uint8_t rx_pipe_number)
{
    if(rx_pipe_number > 5)
        return;
    uint8_t ret = NRF_read_register(NORDIC_EN_RXADDR_REG);
    NRF_write_register(NORDIC_EN_RXADDR_REG, ret | (1<<rx_pipe_number));

}
void NRF_disable_RX_PIPE(uint8_t rx_pipe_number)
{
    if(rx_pipe_number > 5)
```

```
        return;
    uint8_t ret = NRF_read_register(NORDIC_EN_RXADDR_REG);
    NRF_write_register(NORDIC_EN_RXADDR_REG, ret &
(~(1<<rx_pipe_number)));
}

static void NRF_mode_configure(NRF_Mode_t mode, uint8_t rx_pipe_number,
uint8_t addr[5], uint8_t payload_size)
{
      if(mode < 2)
      {
          NRF_radio_disable();
            uint8_t configureRead = NRF_read_config();

            if(mode == NRF_Mode_TX)
            {
                  txconfigured = 1;
                  configureRead &= ~(NORDIC_CONFIG_TX_DS_INT(1));// |
NORDIC_CONFIG_MAX_RT_INT(1));
                  NRF_flush_tx_fifo();
              NRF_write_En_AA(0);
              NRF_write_setup_retry(0);
              NRF_write_TX_ADDR(addr);
              NRF_write_RX_PIPE_ADDR(rx_pipe_number, addr);
              NRF_enable_RX_PIPE(rx_pipe_number);
              NRF_write_register((NORDIC_RX_PW_P0_REG), payload_size);
              NRF_write_config(configureRead | NORDIC_CONFIG_PWR_UP(1));
              DelayMs(2);
            }
            else
            {
                  rxconfigured = 1;
                  configureRead |= NORDIC_CONFIG_PWR_UP(1) |
NORDIC_CONFIG_PRIM_RX(1);
                  configureRead &= ~(NORDIC_CONFIG_RX_DR_INT(1));
                  NRF_flush_rx_fifo();
                  NRF_enable_RX_PIPE(rx_pipe_number);
                  NRF_write_RX_PIPE_ADDR(rx_pipe_number, addr);
                  NRF_write_register((NORDIC_RX_PW_P0_REG +
rx_pipe_number), payload_size);
                  NRF_write_config(configureRead);
                  NRF_radio_enable();
            }

            DelayMs(2);
            printf("NORDIC Configured in %s mode\n", ((mode)?"RX
MODE":"TX MODE"));

      }
      else
      {
            printf("INVALID MODE\n");
      }
}

void NRF_openReadPipe(uint8_t rx_pipe_number, uint8_t rx_addr[5], uint8_t
payload_size)
{
```

```
    NRF_mode_configure(NRF_Mode_RX, rx_pipe_number, rx_addr,
payload_size);
}

void NRF_openWritePipe(uint8_t tx_addr[5])
{
    //NRF_mode_configure(NRF_Mode_TX, 0, tx_addr, 5);
    NRF_mode_configure(NRF_Mode_TX, 0, tx_addr, 32);
}

void NRF_closeWritePipe()
{
    txconfigured = 0;
    uint8_t configureRead = NRF_read_config();
    configureRead |= (NORDIC_CONFIG_TX_DS_INT(1)  |
NORDIC_CONFIG_MAX_RT_INT(1));
    NRF_write_config(configureRead);
    NRF_disable_RX_PIPE(0);
}

void NRF_closeReadPipe(uint8_t rx_pipe_number)
{
    NRF_radio_disable();
    rxconfigured = 0;
    uint8_t configureRead = NRF_read_config();
    configureRead |= NORDIC_CONFIG_RX_DR_INT(1);
    NRF_write_config(configureRead);
    NRF_disable_RX_PIPE(rx_pipe_number);
}

void NRF_write_TXPayload(uint8_t *data, uint8_t len)
{
    NRF_chip_disable();
    NRF_chip_enable();
    SPI_write_byte(NRF_SPI_BUS, NORDIC_W_TXPAYLD_CMD);
    SPI_read_byte(NRF_SPI_BUS); //used to clear the previously value in
the RX FIFO

    SPI_write_packet(NRF_SPI_BUS,data, len);  //loading the FIFO with
data  before enabling the CE pin
    SPI_flushRXFIFO(NRF_SPI_BUS);
    NRF_chip_disable();
}

void NRF_TX_pulse()
{
    NRF_radio_enable();
    //Delay of min 10us
    DelayUs(20);
    NRF_radio_disable();
}

int32_t NRF_transmit_data(uint8_t *data, uint8_t len, uint8_t toRXMode)
{
    int32_t ret = 0;
      if(txconfigured)
      {
          uint8_t configureRead = NRF_read_config();
```

```
          configureRead &= ~NORDIC_CONFIG_PRIM_RX(1);
          NRF_write_config(configureRead);
          //configureRead = NRF_read_config();
          DelayUs(130);

            NRF_radio_disable();

            NRF_write_TXPayload(data, len);

            NRF_TX_pulse();

            printf("Data written");

        uint32_t retry_count = 0;
            if(using_interrupt)
            {
             while(transmitted == 0 && retry_count < 1024)//wait till TX
data is transmitted from FIFO
             {
                 retry_count++;
             }
             if(retry_count == 1024)
             {
                 ret = 0;
                 printf("Data Retry Error\n");
             }
             else
             {
                 ret = len;
                 transmitted = 0; printf("Data Transmitted\n");
             }
            }
            else
            {
             uint8_t status = 0;
             do
             {
                 status = NRF_read_status();
             }while((!((NORDIC_STATUS_TX_DS_MASK |
NORDIC_STATUS_MAX_RT_MASK) & status)) && ++retry_count < 1024);
             if(retry_count > 1023)
             {
                 ret = 0;
                 printf("Data Retry Error\n");
             }
             else
             {
                 NRF_write_status(NORDIC_STATUS_TX_DS_MASK |
NORDIC_STATUS_MAX_RT_MASK);
                 ret = len;
             }
            }
            if(toRXMode)
            {
             configureRead &= ~(NORDIC_CONFIG_PRIM_RX(1));
             NRF_write_config(configureRead);
             NRF_flush_rx_fifo();
             NRF_radio_enable();
```

```
            DelayUs(130);
           }
      }
      else
      {
            printf("TX mode not configured");
        ret = -1;
      }
    return ret;
}

void NRF_read_RXPayload(uint8_t *data, uint8_t len)
{
    NRF_chip_enable();

    SPI_write_byte(NRF_SPI_BUS, NORDIC_R_RXPAYLD_CMD);
    SPI_read_byte(NRF_SPI_BUS);   //used to clear the previously value in
the RX FIFO
    SPI_read_packet(NRF_SPI_BUS,data,len);
    SPI_flush(NRF_SPI_BUS);

    NRF_chip_disable();
}

int32_t NRF_read_data(uint8_t *data, uint8_t len)
{
    int32_t ret = 0;
      if(rxconfigured)
      {
          NRF_radio_enable();
          uint8_t val = NRF_read_fifo_status();
          val = NRF_read_config();
          //TODO: Check how to move forward with this? Call this function
after we know that the data is avail or check with the
          //Status reg if data is available
        uint32_t retry_count = 0;
          if(using_interrupt)
          {
              while(received == 0 && retry_count < 1024)  //wait till RX
data in FIFO
              {
                  //val = NRF_read_fifo_status();//Not needed
                retry_count++;
              }
            if(retry_count > 1023)
            {
                ret = 0;
                printf("Data Retry Error\n");
            }
            else
            {
                  received = 0;
                ret = len;
            }
          }
          else
          {
              uint8_t status = 0;
```

```
              do
              {
                  status = NRF_read_status();
              }while((!(NORDIC_STATUS_RX_DR_MASK & status))&&
++retry_count < 1024);
             if(retry_count > 1023)
             {
                 ret = 0;
                 printf("Data Retry Error\n");
             }
             else
             {
                 NRF_write_status(NORDIC_STATUS_RX_DR_MASK);
                 ret = len;
             }
          }

            printf("Data received");

            NRF_read_RXPayload(data, len);

            printf("Data read");
      }
      else
      {
            printf("RX mode not configured");
        ret = -1;
      }
    return ret;
}

//#define SELF_TEST
#ifdef SELF_TEST

void Nordic_Test()
{
    NRF_moduleInit();
    NRF_moduleSetup(NRF_DR_1Mbps, NRF_PW_LOW);
    DelayMs(100);

    printf( "SPI Initialized\n");
    printf("Nordic Initialized\n");
    printf("Nordic Test\n");
//    NRF_write_status(0);
//    uint8_t sendValue = 0x08;
//    uint8_t readValue = 0;
//    NRF_write_config(sendValue);
//    readValue = NRF_read_config();
//    printf("Recv: 0x%x\n",readValue);
//    if(readValue == sendValue)
//    {
//        printf("Write/Read Config Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    DelayMs(5);
//
```

```
//    NRF_write_register(NORDIC_STATUS_REG,0);
//    sendValue = 44;
//    NRF_write_rf_ch(sendValue);
//    readValue = NRF_read_rf_ch();
//    if(readValue == sendValue)
//    {
//        printf("Write/Read RF CH Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    //sendValue = 0x0F;
//    sendValue = 0x07 ;
//    NRF_write_rf_setup(sendValue);
//    readValue = NRF_read_rf_setup();
//    if(readValue == sendValue)
//    {
//        printf("Write/Read RF Setup Value Matched\n");
//        printf("Sent: 0x%x\n",sendValue);
//        printf("Recv: 0x%x\n",readValue);
//    }
//
//    NRF_write_register(0x03, 3);
//
////    uint8_t sendAddr[5] = {0xBA,0x56,0xBA,0x56,0xBA};
//    uint8_t sendAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
//    printf("TX ADDRESSES SET:
0x%x%x%x%x%x\n",sendAddr[0],sendAddr[1],sendAddr[2],sendAddr[3],sendAddr[
4]);
//    NRF_write_TX_ADDR(sendAddr);
//    uint8_t readAddr[5];
//    NRF_read_TX_ADDR(readAddr);
//    printf("TX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);
//
//    NRF_read_RX_P0_ADDR(readAddr);
//    printf("RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);
//
//    NRF_write_RX_P0_ADDR(sendAddr);
//    NRF_read_RX_P0_ADDR(readAddr);
//    printf("RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);




//    NRF_Mode_t mode = NRF_Mode_RX;
//    printf("Configuring NRF in %d mode",mode);
//    NRF_mode_configure(mode);
//    uint8_t Data[2] = {0};
//    NRF_read_data(Data,2);
//    printf("Nordic Data Recvd: 0x%x, 0x%x", Data[0],Data[1]);

    uint8_t sendAddr[5] = {0xE7,0xE7,0xE7,0xE7,0xE7};
```

```
    NRF_openWritePipe(sendAddr);
    printf("Configuring NRF in TX mode");
    uint8_t readAddr[5];
    NRF_read_TX_ADDR(readAddr);
    logger_log(INFO,"TX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    //NRF_read_RX_P0_ADDR(readAddr);
    logger_log(INFO,"RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    NRF_read_RX_PIPE_ADDR(0, readAddr);
    logger_log(INFO,"RX ADDRESSES GET:
0x%x%x%x%x%x\n",readAddr[0],readAddr[1],readAddr[2],readAddr[3],readAddr[
4]);

    uint8_t Data[5] = {0x55,0xBB,0xBB,0xBB,0xBB};
    NRF_transmit_data(Data,5, false);
    printf("Nordic Data Sent: 0x%x, 0x%x", Data[0],Data[1]);

    printf("Nordic Test End\n");

    NRF_moduleDisable();
}
#endif

void NRF_IntHandler(void *args)
{
    uint8_t NRF_int_reason = NRF_read_status();
    if(NRF_int_reason & NORDIC_STATUS_TX_DS_MASK)
    {
        NRF_write_status(NRF_int_reason | NORDIC_STATUS_TX_DS_MASK);
        transmitted = 1;
        printf("NRF TX Complete\n");
    }
    if(NRF_int_reason & NORDIC_STATUS_RX_DR_MASK)
    {
        NRF_write_status(NRF_int_reason | NORDIC_STATUS_RX_DR_MASK);
        NRF_flush_rx_fifo();
        //TODO: Notification to the handler for the Nordic Data recv task
        user_handler();
        received = 1;
        printf("NRF RX Complete\n");
    }
    if(NRF_int_reason & NORDIC_STATUS_MAX_RT_MASK)
    {
        NRF_write_status(NRF_int_reason | NORDIC_STATUS_MAX_RT_MASK);
        NRF_flush_tx_fifo();
        //TODO: Notification to the handler for the Nordic Data recv task
        user_handler();
        retry_error = 1;
        printf("NRF TX RETRY ERROR\n");
    }
}

#endif
```

```
/**
 * @brief
 *
 * @file apds9301_sensor.c
 * @author Gunj Manseta
 * @date 2018-03-14
 */

#include <math.h>
#include "apds9301_sensor.h"
#include"my_i2c.h"
#include "error_data.h"
#include <string.h>

#define APDS9301_SENSOR_ID (0x50)

int APDS9301_test()
{
    uint8_t data;
    int ret = APDS9301_readID(&data);
    if(ret == 0)
    {
        (APDS9301_SENSOR_ID == (data & 0xF0)) ? 0 : (ret = -1);
    }

    return ret;

}

int APDS9301_write_ThLow(uint16_t thlow)
{
    int ret = I2Cmaster_write_word(APDS9301_SLAVE_ADDR,
APDS9301_INT_TH_LL_REG | APDS9301_CMD_WORD_EN, thlow, 0);
    return ret;
}

int APDS9301_write_ThHigh(uint16_t thhigh)
{
    int ret = I2Cmaster_write_word(APDS9301_SLAVE_ADDR,
APDS9301_INT_TH_HL_REG | APDS9301_CMD_WORD_EN , thhigh, 0);
    return ret;
}

int APDS9301_read_ThLow(uint16_t *thlow)
{
    int ret = I2Cmaster_read_bytes(APDS9301_SLAVE_ADDR,
APDS9301_INT_TH_LL_REG, (uint8_t*)thlow, sizeof(thlow));
    return ret;
}

int APDS9301_read_ThHigh(uint16_t *thhigh)
{
    int ret = I2Cmaster_read_bytes(APDS9301_SLAVE_ADDR,
APDS9301_INT_TH_HL_REG, (uint8_t*)thhigh, sizeof(thhigh));
    return ret;
}
```

```
int APDS9301_readControlReg(uint8_t *ctrl_reg)
{
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_CTRL_REG,
ctrl_reg);
    return ret;
}

int APDS9301_mode_highGain()
{
    uint8_t data;
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR,
APDS9301_TIMING_REG, &data);
    if(ret != 0)
        return ret;

    data |= (uint8_t)APDS9301_TIMING_GAIN;

    ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR, APDS9301_TIMING_REG,
data);

    return ret;
}
int APDS9301_mode_lowGain_default()
{
    uint8_t data;
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR,
APDS9301_TIMING_REG, &data);
    if(ret != 0)
        return ret;

    data &= ~(uint8_t)APDS9301_TIMING_GAIN;

    ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR, APDS9301_TIMING_REG,
data);

    return ret;

}

int APDS9301_mode_interrupt(uint8_t enable)
{
    uint8_t data;
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR,
APDS9301_INT_CTRL_REG, &data);
    if(ret != 0)
        return ret;

    if(enable)
        data |= (uint8_t)APDS9301_INTCTRL_IEN;
    else
        data &= ~(uint8_t)APDS9301_INTCTRL_IEN;

    ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR,
APDS9301_INT_CTRL_REG, data);

    return ret;
}
```

```
int APDS9301_clearPendingInterrupt()
{
    int ret = I2Cmaster_write(APDS9301_SLAVE_ADDR, APDS9301_CMD_REG |
APDS9301_CMD_INT_CLEAR);
    return ret;
}

int APDS9301_mode_manualcontrol(uint8_t on)
{
    uint8_t data;
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR,
APDS9301_TIMING_REG, &data);
    if(ret != 0)
        return ret;

    data &= ~(uint8_t)APDS9301_TIMING_MANUAL(1);
    data |= (uint8_t)APDS9301_TIMING_MANUAL(on);

    ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR, APDS9301_TIMING_REG,
data);

    return ret;
}

int APDS9301_mode_integrationTime(uint8_t x)
{
    uint8_t data;
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR,
APDS9301_TIMING_REG, &data);
    if(ret != 0)
        return ret;

    data &= ~(uint8_t)APDS9301_TIMING_INTEG(3);
    data |= (uint8_t)APDS9301_TIMING_INTEG(x);

    ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR, APDS9301_TIMING_REG,
data);

    return ret;
}


int APDS9301_poweron()
{
    int ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR,
APDS9301_CTRL_REG, APDS9301_CTRL_POWERON);
    return ret;
}
int APDS9301_powerdown()
{
    int ret = I2Cmaster_write_byte(APDS9301_SLAVE_ADDR,
APDS9301_CTRL_REG, APDS9301_CTRL_POWEROFF);
    return ret;
}
int APDS9301_readID(uint8_t *id)
{
    int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_ID_REG,
id);
```

```
    return ret;
}

int APDS9301_readCh0(uint16_t *ch0_data)
{
    int ret;
    uint8_t ch0_data_L, ch0_data_H;
    ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_CH0_DATALOW,
&ch0_data_L);
    if(ret)
        return ret;

    ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_CH0_DATAHIGH,
&ch0_data_H);

    if(!ret)
        *ch0_data = (ch0_data_H << 8) | ch0_data_L ;

    return ret;
}

int APDS9301_readCh1(uint16_t *ch1_data)
{
    int ret;
    uint8_t ch1_data_L, ch1_data_H;
    ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_CH1_DATALOW,
&ch1_data_L);
    if(ret)
        return ret;

    ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, APDS9301_CH1_DATAHIGH,
&ch1_data_H);

    if(!ret)
        *ch1_data = (ch1_data_H << 8) | ch1_data_L;

    return ret;

}

float APDS9301_getLux()
{
    float ratio, lux = -1;
    uint16_t Ch0, Ch1;

    int ret  = APDS9301_readCh0(&Ch0);
    if(ret)
        return lux;

    ret  = APDS9301_readCh1(&Ch1);
    if(ret)
        return lux;

    if(Ch0 != 0)
        ratio = (float)Ch1/(float)Ch0;
    else
        ratio = 0;
```

```
    //Calculate LUX - calculations are referred from the Sensor datasheet
      if (ratio > 0 && ratio <= 0.50)
      {
            lux = 0.0304*Ch0 - 0.062*Ch0*(pow(ratio, 1.4));
      }
      else if (ratio > 0.50 && ratio <= 0.61)
      {
            lux = 0.0224*Ch0 - 0.031*Ch1;
      }
      else if (ratio > 0.61 && ratio <= 0.80)
      {
            lux = 0.0128*Ch0 - 0.0153*Ch1;
      }
      else if (ratio > 0.80 && ratio <= 1.30)
      {
            lux = 0.00146*Ch0 - 0.00112*Ch1;
      }
      else if (ratio > 1.30)
      {
            lux = 0;
    }

    return lux;
}


uint8_t* APDS9301_memDump()
{
    uint8_t *memdump = (uint8_t*)malloc(15*sizeof(uint8_t));
    memset(memdump, 0 , 15);

    for(uint8_t i = 0 ; i < 0x10; i++)
    {
        if(i == 0x7 ||  i == 0x9 || i == 0xB)
            continue;

        int ret = I2Cmaster_read_byte(APDS9301_SLAVE_ADDR, i |
APDS9301_CMD_REG, memdump+i);
        if(ret != 0 ) continue;
    }

    return memdump;
}

int APDS9301__setmode_allDefault()
{
    int ret = APDS9301_mode_lowGain_default();
    if(ret) return ret;
    ret = APDS9301_mode_integrationTime2_default();
    if(ret) return ret;
    ret = APDS9301_mode_interruptDisable_default();
    if(ret) return ret;
    ret = APDS9301_mode_manualcontrolOFF_default();
    return ret;
}/**
 * @brief
 *
 * @file comm_sender_task.c
```

```
 * @author Gunj Manseta
 * @date 2018-04-26
 */

#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>
#include <signal.h>

#include "main_task.h"
#include "common_helper.h"
#include "comm_sender_task.h"
#include "communication_object.h"
#include "communication_interface.h"

#define MQ_COMM_SENDER_TASK_NAME "/comm_sender_task_queue"


static mqd_t comm_sender_task_q;

mqd_t getHandle_CommSenderTaskQueue()
{
    return comm_sender_task_q;
}

/* Create a queue to get the request and process that. According to the
msg id, create the packet and send it */
int comm_sender_task_queue_init()
{
    struct mq_attr comm_sender_taskQ_attr = {
        .mq_msgsize = sizeof(COMM_MSG_T),
        .mq_maxmsg = 128,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_COMM_SENDER_TASK_NAME);
    comm_sender_task_q = mq_open(MQ_COMM_SENDER_TASK_NAME, O_CREAT |
O_RDWR, 0666, &comm_sender_taskQ_attr);

    return comm_sender_task_q;
}

void comm_sender_task_processMsg()
{
    int ret,prio;
    COMM_MSG_T queueData = {0};
    DEFINE_MAINTASK_STRUCT(maintaskRsp,MT_MSG_STATUS_RSP,LOGGER_TASK_ID);
    //struct timespec recv_timeout = {0};
    uint8_t continue_flag= 1;
    while(continue_flag)
    {
        memset(&queueData,0,sizeof(queueData));
        // clock_gettime(CLOCK_REALTIME, &recv_timeout);
        // recv_timeout.tv_sec += 3;
```

```
        // ret =
mq_timedreceive(loggertask_q,(char*)&(queueData),sizeof(queueData),&prio,
&recv_timeout);
        ret =
mq_receive(comm_sender_task_q,(char*)&(queueData),sizeof(queueData),&prio
);
        // if(ERR == ret && ETIMEDOUT == errno)
        // {
        //     //LOG_STDOUT(ERROR "MQ_RECV
TIMEOUT:%s\n",strerror(errno));
        //     continue;
        // }
        if(ERR == ret )
        {
            LOG_STDOUT(ERROR "MQ_RECV:%s\n",strerror(errno));
            continue;
        }
        switch(queueData.msg_id)
        {
            case 0xFF:
                (queueData.dst_brd_id == BBG_BOARD_ID) ? continue_flag =
0 :0;
                break;
            case MSG_ID_GET_SENSOR_STATUS:
                if(sizeof(queueData) != COMM_SEND(&queueData))
                    LOG_STDOUT(WARNING "COMM SEND NO HOST\n");
                break;
            case MSG_ID_GET_SENSOR_INFO:
                if(sizeof(queueData) != COMM_SEND(&queueData))
                    LOG_STDOUT(WARNING "COMM SEND NO HOST\n");
                break;
            case MSG_ID_GET_CLIENT_INFO_BOARD_TYPE:
                if(sizeof(queueData) != COMM_SEND(&queueData))
                    LOG_STDOUT(WARNING "COMM SEND NO HOST\n");
                break;
            case MSG_ID_GET_CLIENT_INFO_UID:
                if(sizeof(queueData) != COMM_SEND(&queueData))
                    LOG_STDOUT(WARNING "COMM SEND NO HOST\n");
                break;
            case MSG_ID_GET_CLIENT_INFO_CODE_VERSION:
                if(sizeof(queueData) != COMM_SEND(&queueData))
                    LOG_STDOUT(WARNING "COMM SEND NO HOST\n");
                break;
            default:
                LOG_STDOUT(INFO "Invalid msg id\n");
                break;

        }
    }
}

/* Create the entry function */
void* comm_sender_task_callback(void *threadparam)
{
    LOG_STDOUT(INFO "COMM SENDER STARTED\n");
    int ret = comm_sender_task_queue_init();
    if(ERR == ret)
    {
```

```
        LOG_STDOUT(ERROR "COMM SENDER INIT%s\n",strerror(errno));
        //exit(ERR);
        goto EXIT;
    }

    UART_FD_T fd = COMM_INIT();
    if(fd < 0)
        goto EXIT_COMM;

    LOG_STDOUT(INFO "COMM SENDER INIT DONE\n");
    pthread_barrier_wait(&tasks_barrier);

    comm_sender_task_processMsg();

    COMM_DEINIT(fd);

EXIT_COMM:
    mq_close(comm_sender_task_q);
EXIT:
    LOG_STDOUT(INFO "COMM SENDER TASK EXIT\n");
    return SUCCESS;

}/**
 * @brief
 *
 * @file my_uart.c
 * @author Gunj Manseta
 * @date 2018-04-23
 */

#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <termios.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>

#include "my_uart.h"
#include "communication_object.h"

#define BAUD_115200     B115200
#define BAUD_9921600    B921600

#define BAUDRATE    BAUD_9921600

const char *const COMPORT[5] =
{"","/dev/ttyS1","/dev/ttyS2","/dev/ttyS3","/dev/ttyS4"};

/* These functions are specific to the ttyS4*/
#define UART_DEV "/dev/ttyS4" //Beaglebone Green serial port


//caching the old tio config to keep it back as it was
static struct termios old_tio_config;
```

```
static int32_t opened = 0;
static int internal_fd = -1;

UART_FD_T UART_Open(COM_PORT com_port)
{
    //Not supporing other com ports as I dont have time to handle open
close and release for all the com ports
    if(com_port != COM_PORT4)
        return -1;

    if(opened > 0 && internal_fd != -1)
    {
        if(opened < 64)
            return internal_fd;
        else
            return -1;
    }

    int fd;
    struct termios tio_config;
    char buf[255];

    fd = open(COMPORT[COM_PORT4], O_RDWR | O_NOCTTY | O_SYNC );
    if (fd < 0)
    {
        perror(UART_DEV);
        return -1;
        //exit(-1);
    }

    tcgetattr(fd, &old_tio_config);

    bzero(&tio_config, sizeof(tio_config));

    tio_config.c_cflag = BAUDRATE | CS8 | CLOCAL | CREAD;
    cfmakeraw(&tio_config);
    tio_config.c_cc[VMIN]  = 0;
    tio_config.c_cc[VTIME] = 1;

    tcflush(fd, TCIFLUSH);
    tcsetattr(fd, TCSANOW, &tio_config);

    internal_fd = fd;
    opened++;
    return fd;
}

void UART_Release(UART_FD_T fd)
{
    tcsetattr(fd, TCSANOW, &old_tio_config);
    close(fd);
}

void UART_Close(UART_FD_T fd)
{
    opened--;
    if(opened > 0)
    {
```

```
        //dummy close
    }
    else
    {
        if(fd == internal_fd)
        {
            UART_Release(fd);
            internal_fd = -1;
        }
    }
}

void UART_flush()
{
    (internal_fd > 0) ? tcflush(internal_fd, TCIFLUSH): 0;
}

int32_t UART_putchar(char c)
{
    return write(internal_fd, &c, 1);
}

int32_t UART_putRAW(void *object, size_t len)
{
    if(internal_fd < 0)
        return -1;

    int32_t ret = 0, retry = 0, i =0;
    do
    {
        ret = write(internal_fd, object+i, len-i);
        i += ret;
        retry++;
    }while(ret > -1 && i < len && retry < 16);
    //printf("I:%d RET: %dRETRY: %d\n",i, ret, retry);
    return ret;
}

int32_t UART_putstr(const char* str)
{
    return UART_putRAW((void*)str, strlen(str));
}

int32_t UART_printf(const char *fmt, ...)
{
    //this function is not needed
}

int32_t UART_read(void *object, size_t len)
{
    if(internal_fd < 0)
        return -1;

    //printf("IN UART READ\n");
    int ret = 0, retry = 0, i =0;
    do
    {
        ret = read(internal_fd, object+i, len-i);
```

```
        i +=ret;
        retry++;
        //printf("IN: i = %d ret: %d try: %d\n",i, ret, retry);
    }while(ret > -1 && i < len && retry < 16);

    //(retry>=1) ? printf("i = %d ret: %d try: %d\n",i, ret, retry): 0;
    return ret;
}

int32_t UART_dataAvailable(uint32_t time_ms)
{
    if(internal_fd < 0)
        return -1;

    struct timeval timeout;

    //dont wait
    if (time_ms == 0)
    {
        timeout.tv_sec = 0;
        timeout.tv_usec = 0;
    }
    else
    {
        timeout.tv_sec = time_ms / 1000;
        timeout.tv_usec = (time_ms % 1000) * 1000;
    }

    fd_set readfds;

    FD_ZERO(&readfds);
    FD_SET(internal_fd, &readfds);

    if(select(internal_fd + 1, &readfds, NULL, NULL, &timeout) > 0)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

#ifdef SELF_TEST
#include <errno.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

volatile sig_atomic_t flag = 1;
#define SIGNAL "SIGNAL"

void signal_handler(int signal)
{
    switch (signal)
      {
            case SIGINT:
                 printf(SIGNAL "SIGINT signal.\n");
```

```
                flag = 0;
            break;
            case SIGTERM:
                printf(SIGNAL "SIGTERM signal.\n");
            flag = 0;
                break;
            case SIGTSTP:
                printf(SIGNAL "SIGTSTP signal.\n");
            flag = 0;
                break;
            default:
                printf(SIGNAL "Invalid signal.\n");
                break;
      }
}

int main()
{
    signal(SIGINT, signal_handler);
    signal(SIGTERM, signal_handler);
    signal(SIGTSTP, signal_handler);
    printf("SIZE: %u\n",sizeof(COMM_MSG_T));
    int fd = UART_Open(COM_PORT4);
    if(fd == -1)
    {
        printf("UART open ERROR");
        return fd;
    }
    int ret = 0;

    #ifdef UART_COMM_TEST
    #ifdef LOOPBACK
      COMM_MSG_T comm_msg =
    {
        .ID = HEARTBEAT | (0x55<<24),
        .payload = "SELF CHECK",
    };
    comm_msg.checksum = getCheckSum(&comm_msg);


      printf("SIZEOF: %u\n",sizeof(comm_msg));
    printf("Sending:\nBOARDID: 0x%01x ID:%u, PAYLOAD:%s,
CHECKSUM:%u\n",GET_BOARD_UID_FROM_LOG_ID(comm_msg.ID),GET_LOG_ID_FROM_LOG
_ID(comm_msg.ID),comm_msg.payload,comm_msg.checksum);

    ret = UART_putRAW(&comm_msg,sizeof(comm_msg));
    if(ret == -1)
        printf("Serial Write Error: %s\n",strerror(errno));
    else
        printf("Bytes sent: %d\n",ret);

    #endif

    COMM_MSG_T recv_comm_msg = {0};

    #ifndef LOOPBACK
    while(flag)
    {
```

```
    #endif
    ret = UART_read(&recv_comm_msg,sizeof(recv_comm_msg));
    if(ret == -1)
        printf("Serial Read Error: %s\n",strerror(errno));
    else
        printf("Bytes recvd: %d\n",ret);

    uint16_t check = getCheckSum(&recv_comm_msg);
    printf("\n*******\n     \
    SRCID:%u, SRC_BRDID:%u, DST_ID:%u, MSGID:%u\n   \
    MSG:%s\n    \
    Checksum:%u ?= %u\n********\n",   \
    recv_comm_msg.src_id, recv_comm_msg.src_brd_id,
recv_comm_msg.dst_id,recv_comm_msg.msg_id,
recv_comm_msg.message,recv_comm_msg.checksum, check );
    #ifndef LOOPBACK
    }
    #endif
    #endif

    #ifdef CAMERA_TEST
    uint8_t buffer[3600] = {0};
    uint8_t temp = 0, temp_last = 0;
    uint32_t len = 0;
    char buff[4] = {0};
    int i = 0;
    uint8_t done = 0;
    uint32_t retry = 0;
    struct packet pack = {0};
    uint8_t getpix = 0;
    uint8_t header = 0;
        while(1)
        {
        int32_t ret = UART_read((uint8_t*)&getpix,1);
        /* Some error */
        //printf("RET:%d Retry:%d\n",ret,retrycount);
        if(ret == -1)
        {
            /* LOG error */
            printf("COMM RECV\n");
        }
        else if(ret > 0 && (getpix == 0xFF || header == 1) )
        {
            if(ret == 1)
            {
                if(getpix == 0xFF)
                    header = 1;
                buffer[i] = getpix;
                printf("0x%x\n",buffer[i]);
                if(temp_last == 0xFF && buffer[i] == 0xD9)
                {
                    printf("EOF found\n");
                    done = 1;
                    break;
                }
                temp_last  = buffer[i];
                i++;
            }
```

```
        }
        else
        {
            retry++;
            if(retry > 54)
            {
                printf("COnnected?\n");
            }
        }
        }

    if(done)
    {
        char newFilename[40] = {0};

snprintf(newFilename,sizeof(newFilename),"%u_%s.%s",image_count,"image","
jpg");
        FILE *fp = fopen(newFilename, "wb");
        fwrite(buffer,i,1,fp);
        fclose(fp);
    }

    #endif

    tcflush(fd, TCIFLUSH);
    UART_Close(fd);

      return 0;
}
#endif

/**
 * @brief
 *
 * @file socket_task.c
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#include <sys/socket.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <errno.h>
#include <arpa/inet.h>

#include "socket_task.h"
#include "logger_task.h"
#include "main_task.h"
#include "error_data.h"
#include "common_helper.h"
#include "sensor_common_object.h"
#include "light_sensor_task.h"
#include "temperature_sensor_task.h"
#include "comm_sender_task.h"

#define SERVER_PORT    3000
```

```
#define SERVER_IP       "127.0.0.1"
#define MAX_CONNECTIONS 5

sig_atomic_t socketTask_continue = 1;

/**
 * @brief
 *
 * @param req_in
 * @return REMOTE_RESPONSE_T
 */
REMOTE_RESPONSE_T processRemoteRequest(REMOTE_REQUEST_T req_in);

/**
 * @brief
 *
 * @param sigval
 */
static void timer_handler_sendSTAliveMSG(union sigval sig)
{
    pthread_mutex_lock(&aliveState_lock);
      aliveStatus[SOCKET_TASK_ID]++;
      pthread_mutex_unlock(&aliveState_lock);
}

/**
 * @brief
 *
 * @param server_socket
 * @return int
 */
int socket_task_init(int server_socket)
{
    int option = 1;
      struct sockaddr_in addr;

      if((server_socket = socket(AF_INET,SOCK_STREAM,0)) == 0)
      {
            LOG_STDOUT(ERROR "Socket Creation:%s\n",strerror(errno));
            return ERR;
      }

      LOG_STDOUT(INFO "Socket Created\n");

      if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEPORT |
SO_REUSEADDR, &(option), sizeof(option)))
    {
        LOG_STDOUT(ERROR "Cannot Set socket
options:%s\n",strerror(errno));
        return ERR;
    }
      /*Setting up the sockaddr_in structure */
      addr.sin_family = AF_INET;
      //addr.sin_addr.s_addr = inet_addr(SERVER_IP);
      addr.sin_addr.s_addr = INADDR_ANY;//Using local loopback
      addr.sin_port = htons(SERVER_PORT);

      if((bind(server_socket,(struct sockaddr*)&addr, sizeof(addr))) < 0)
```

```
      {
            LOG_STDOUT(ERROR "Cannot bind the
socket:%s\n",strerror(errno));
            return ERR;
      }

      LOG_STDOUT(INFO "Socket binded\n");

    if(listen(server_socket,MAX_CONNECTIONS) < 0)
      {
            LOG_STDOUT(ERROR "Cannot listen:%s\n",strerror(errno));
            return ERR;
      }

    LOG_STDOUT(INFO "Socket started listening on IP:%s
PORT:%d\n",SERVER_IP,SERVER_PORT);

    return server_socket;
}

float getDistanceData_cm();

void* socket_task_callback(void* threadparam)
{
    int server_socket,accepted_socket, option = 1;
    struct sockaddr_in peer_addr;
      int addrLen = sizeof(peer_addr);
    LOG_STDOUT(INFO "SOCKET TASK STARTED\n");

    server_socket  = socket_task_init(server_socket);
    if(ERR == server_socket)
    {
        LOG_STDOUT(ERROR "Socket task init failed.\n");
        exit(ERR);
    }


      LOG_STDOUT(INFO "SOCKET TASK INIT COMPLETED\n");
      pthread_barrier_wait(&tasks_barrier);

      DEFINE_LOG_STRUCT(logData,LT_MSG_LOG,SOCKET_TASK_ID);

      while(socketTask_continue)
      {
            POST_MESSAGE_LOGTASK(&logData,"Accepting connections...\n");
            accepted_socket = accept(server_socket, (struct
sockaddr*)&peer_addr,(socklen_t*)&addrLen);
            if(accepted_socket < 0)
            {
                  LOG_STDOUT(ERROR "Cannot accept:%s\n",strerror(errno));
                  POST_MESSAGE_LOGTASK(&logData,ERROR "Cannot
accept:%s\n",strerror(errno));
                  continue;
            }

            char peer_IP[20] = {0};
            POST_MESSAGE_LOGTASK(&logData,INFO "Conn accepted. Peer Add:
%s\n",inet_ntop(AF_INET, &peer_addr.sin_addr, peer_IP, sizeof(peer_IP)));
```

```
            LOG_STDOUT(INFO "Connection accepted from peer Addr:
%s\n",inet_ntop(AF_INET, &peer_addr.sin_addr, peer_IP, sizeof(peer_IP)));

            while(socketTask_continue)
            {
                  REMOTE_REQUEST_T req_in = {0};
                  REMOTE_RESPONSE_T rsp_out = {0};
                  int nbytes = 0;
                  do{
                        nbytes = recv(accepted_socket,
(((char*)&(req_in))+nbytes), sizeof(req_in), 0);
                  }while(nbytes < sizeof(req_in) && nbytes != -1);
                  if(nbytes == -1)
                  {
                        LOG_STDOUT(INFO "SOME ERROR ON SOCKET\n");
                        continue;
                  }
                  LOG_STDOUT(INFO "--CLIENT REQUEST: bytes:%d
ID:%d\n",nbytes,req_in.request_id);
                  POST_MESSAGE_LOGTASK(&logData,INFO "--CLIENT REQUEST:
bytes:%d ID:%d\n",nbytes,req_in.request_id);
                  rsp_out = processRemoteRequest(req_in);


                  nbytes = send(accepted_socket , (char*)&rsp_out ,
sizeof(rsp_out), 0 );
                  if(nbytes < sizeof(rsp_out))
                  {
                        LOG_STDOUT(ERROR "Cannot send complete data\n");
                        POST_MESSAGE_LOGTASK(&logData,ERROR "Cannot send
complete data\n");
                        break;
                  }

                  LOG_STDOUT(INFO "Number of bytes sent: %d\n",nbytes);
                  POST_MESSAGE_LOGTASK(&logData,INFO "Number of bytes
sent: %d\n",nbytes);

                  if(rsp_out.rsp_id == CONN_CLOSE_RSP)
                        {break;}
                  if(rsp_out.rsp_id == CONN_KILL_APP_RSP)
                        { socketTask_continue = 0; break;}
            }

        /* Create a new thread to handle the connection and go back to
accepting */
        close(accepted_socket);
            LOG_STDOUT(INFO "Socket Closed\n");
            POST_MESSAGE_LOGTASK(&logData,INFO "Socket Closed\n");
        /* Think of a mechanism to close this socket task as there is a
while(1) loop */

    }

}

REMOTE_RESPONSE_T processRemoteRequest(REMOTE_REQUEST_T req_in)
{
```

```
      REMOTE_RESPONSE_T rsp_out = {0};
      switch(req_in.request_id)
    {
        case(GET_FUNC):
                rsp_out.rsp_id=GET_FUNC;
            strncpy(rsp_out.metadata,"GET_TEMP DAY_NIGHT\n",
sizeof(rsp_out.metadata));
            break;
        case(GET_TEMP_C):
                rsp_out.rsp_id=GET_TEMP_C;
                rsp_out.data.floatingData = getTempTask_temperature();
            LOG_STDOUT(INFO "REMOTE REQUEST
GET_TEMP_C:.03%f\n",rsp_out.data.floatingData);
            break;
        case(GET_TEMP_F):
                rsp_out.rsp_id=GET_TEMP_F;
                rsp_out.data.floatingData = getTempTask_temperature();
                rsp_out.data.floatingData = (rsp_out.data.floatingData *
1.8) + 32;
            LOG_STDOUT(INFO "REMOTE REQUEST
GET_TEMP_F:%.03f\n",rsp_out.data.floatingData);
            break;
        case(GET_TEMP_K):
                rsp_out.rsp_id=GET_TEMP_K;
                rsp_out.data.floatingData = getTempTask_temperature() +
273.15;
            LOG_STDOUT(INFO "REMOTE REQUEST
GET_TEMP_K:%.03f\n",rsp_out.data.floatingData);
            break;
        case(GET_LUX):
                rsp_out.rsp_id=GET_LUX;
                rsp_out.data.floatingData = getLightTask_lux();
            LOG_STDOUT(INFO "REMOTE REQUEST
GET_LUX:%.03f\n",rsp_out.data.floatingData);
            break;
        case(GET_DAY_NIGHT):
                rsp_out.rsp_id=GET_DAY_NIGHT;
                rsp_out.data.isNight = getLightTask_state();
            LOG_STDOUT(INFO "REMOTE REQUEST GET_DAY_NIGHT\n");
            break;
           case(GET_DISTANCE_CM):
                LOG_STDOUT(INFO "REMOTE REQUEST GET_DISTANCE_CM\n");
                rsp_out.rsp_id=GET_DISTANCE_CM;
                rsp_out.data.floatingData = getDistanceData_cm();
                LOG_STDOUT(INFO "SERVER RESPONSE GOT_DISTANCE_CM\n");
                break;
           case(GET_DISTANCE_M):
                LOG_STDOUT(INFO "REMOTE REQUEST GET_DISTANCE_M\n");
                rsp_out.rsp_id=GET_DISTANCE_M;
                rsp_out.data.floatingData = (getDistanceData_cm()/100);
                LOG_STDOUT(INFO "SERVER RESPONSE GOT_DISTANCE_M\n");
                break;
      case(CONN_CLOSE_REQ):
                rsp_out.rsp_id=CONN_CLOSE_RSP;
                LOG_STDOUT(INFO "REMOTE REQUEST CLOSE CONNECTION\n");
             break;
           case(CONN_KILL_APP_REQ):
                rsp_out.rsp_id=CONN_KILL_APP_RSP;
```

```
                LOG_STDOUT(INFO "REMOTE REQUEST KILL APP\n");

      DEFINE_TEMP_STRUCT(tempstruct,TEMP_MSG_TASK_EXIT,MAIN_TASK_ID)
                POST_MESSAGE_TEMPERATURETASK_EXIT(&tempstruct);
            break;
        default:
                rsp_out.rsp_id=GET_FUNC;
                LOG_STDOUT(INFO "REMOTE REQUEST INVALID\n");
            break;

    }
      return rsp_out;
}

uint8_t gotDistance = 0;
COMM_MSG_T socket_comm_msg = {0};

float getDistanceData_cm()
{
      /* Send the request to comm send task */
      send_GET_DISTANCE(TIVA_BOARD1_ID,BBG_SOCKET_MODULE);
      /* Wait for notification event from dispatcher */
      while(gotDistance == 0);
      gotDistance = 0;
      /* Get the comm object and return it. */
      float data = socket_comm_msg.data.distance_cm;
      memset(&socket_comm_msg,0,sizeof(socket_comm_msg));
      return data;
}/**
 * @brief
 *
 * @file my_i2c.c
 * @author Gunj Manseta
 * @date 2018-03-13
 */

#include <errno.h>
#include <string.h>
#include "my_i2c.h"


static I2C_MASTER_HANDLE_T *internal_master_handle = NULL;
static pthread_mutex_t init_destroy_lock = PTHREAD_MUTEX_INITIALIZER;

void printErrorCode(int errorCode)
{
    //#define VERBOSE
    #ifdef VERBOSE
    mraa_result_print(errorCode);
    #endif
}

int I2Cmaster_Init(I2C_MASTER_HANDLE_T *handle)
{
    int ret = 0;
    if(pthread_mutex_lock(&init_destroy_lock))
        return -1;
    if(NULL != internal_master_handle)
```

```
    {
        handle = internal_master_handle;
        ret =  0;
    }
    else if(handle)
    {
        handle->i2c_context = mraa_i2c_init_raw(BB_I2C_BUS_2);
        /* internal i2c context failed to init */
        if(NULL == handle->i2c_context)
        {
            internal_master_handle = NULL;
            ret = -1;
        }
        /* If spinlock init fails */
        else if( -1 == pthread_spin_init(&handle->handle_lock,
PTHREAD_PROCESS_PRIVATE))
        {
            internal_master_handle = NULL;
            mraa_i2c_stop(handle->i2c_context);
            ret = -1;
        }
        /* Everyting goes as expected */
        else
        {
            internal_master_handle = handle;
            ret = 0;
        }
    }
    /* If handle is null */
    else
        ret = -1;

    if(pthread_mutex_unlock(&init_destroy_lock))
        return -1;

    return ret;
}

int I2Cmaster_Destroy(I2C_MASTER_HANDLE_T *handle)
{
    int ret;
    if(pthread_mutex_lock(&init_destroy_lock))
        return -1;

    /* If the input handle is not null and the input initialized handle
should match the internal handle */
    if(NULL != handle && internal_master_handle == handle && NULL !=
internal_master_handle)
    {
        ret = mraa_i2c_stop(handle->i2c_context);
        if(ret == 0)
        {
            static int timeout = 5000;
            do{
                ret = pthread_spin_destroy(&handle->handle_lock);
                timeout--;
            }while(EBUSY == ret && timeout > 0);
```

```
            if(ret == 0)
            {
                internal_master_handle = NULL;
            }
        }
    }
    else if(NULL == internal_master_handle)
    {
        ret = 0;
    }
    else
    {
        ret = -1;
    }

    if(pthread_mutex_unlock(&init_destroy_lock))
        return -1;

    return ret;
}

I2C_MASTER_HANDLE_T* getMasterI2C_handle()
{
    return internal_master_handle;
}

int I2Cmaster_write_byte(uint8_t slave_addr, uint8_t reg_addr, uint8_t
data)
{
    if(NULL == internal_master_handle)
    {
        return -1;
    }

    pthread_spin_lock(&internal_master_handle->handle_lock);

    mraa_result_t ret =  mraa_i2c_address(internal_master_handle-
>i2c_context, slave_addr);
    if(0 == ret)
    {
        ret = mraa_i2c_write_byte_data(internal_master_handle-
>i2c_context, data, reg_addr);
    }

    pthread_spin_unlock(&internal_master_handle->handle_lock);

    return ret;
}

int I2Cmaster_write(uint8_t slave_addr, uint8_t reg_addr)
{
    if(NULL == internal_master_handle)
    {
        return -1;
    }

    pthread_spin_lock(&internal_master_handle->handle_lock);
```

```
    mraa_result_t ret =  mraa_i2c_address(internal_master_handle-
>i2c_context, slave_addr);
    if(0 == ret)
    {
        ret = mraa_i2c_write_byte(internal_master_handle->i2c_context,
reg_addr);
    }

    pthread_spin_unlock(&internal_master_handle->handle_lock);

    return ret;
}


int I2Cmaster_write_word(uint8_t slave_addr, uint8_t reg_addr, uint16_t
data, uint8_t lsb_first)
{
    if(NULL == internal_master_handle)
    {
        return -1;
    }

    if(lsb_first)
    {
        data = ( ((data & 0xF0)>>4)| ((data &0x0F)<<4) );
    }

    pthread_spin_lock(&internal_master_handle->handle_lock);

    mraa_result_t ret =  mraa_i2c_address(internal_master_handle-
>i2c_context, slave_addr);
    if(0 == ret)
    {
        ret = mraa_i2c_write_word_data(internal_master_handle-
>i2c_context, data, reg_addr);
    }

    pthread_spin_unlock(&internal_master_handle->handle_lock);

    return ret;

}


int I2Cmaster_read_byte(uint8_t slave_addr, uint8_t reg_addr, uint8_t
*data)
{
    if(NULL == internal_master_handle)
    {
        return -1;
    }

    pthread_spin_lock(&internal_master_handle->handle_lock);

    mraa_result_t ret =  mraa_i2c_address(internal_master_handle-
>i2c_context, slave_addr);
    if(0 == ret)
    {
```

```
        ret = mraa_i2c_read_byte_data(internal_master_handle-
>i2c_context, reg_addr);
        (ret != -1) ? *data  = ret, ret = 0 : 0;
    }

    pthread_spin_unlock(&internal_master_handle->handle_lock);

    return ret;

}

int I2Cmaster_read_bytes(uint8_t slave_addr, uint8_t reg_addr, uint8_t
*data, size_t len)
{
    if(NULL == internal_master_handle)
    {
        return -1;
    }

    pthread_spin_lock(&internal_master_handle->handle_lock);

    mraa_result_t ret =  mraa_i2c_address(internal_master_handle-
>i2c_context, slave_addr);
    if(0 == ret)
    {
        ret = mraa_i2c_read_bytes_data(internal_master_handle-
>i2c_context, reg_addr, data , len);
        (ret == len) ? ret = 0 : 0;
    }

    pthread_spin_unlock(&internal_master_handle->handle_lock);

    return ret;
}

/**
 * @brief
 *
 * @file main_task.c
 * @author Gunj Manseta
 * @date 2018-03-09
 */

#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>
#include "main_task.h"
#include "error_data.h"
#include "logger_task.h"
#include "socket_task.h"
#include "light_sensor_task.h"
#include "temperature_sensor_task.h"
#include "my_signals.h"
#include "posixTimer.h"
#include "common_helper.h"
```

```
#include "readConfiguration.h"
#include "comm_recv_task.h"
#include "comm_sender_task.h"
#include "dispatcher_task.h"
#include "BB_Led.h"


#define MQ_MAINTASK_NAME "/maintask_queue"

volatile int timeoutflag;
volatile sig_atomic_t signal_exit;
volatile int aliveStatus[NUM_CHILD_THREADS] = {0};
extern volatile sig_atomic_t comm_recv_task_exit;

void* (*thread_callbacks[NUM_CHILD_THREADS])(void *) =
{
    logger_task_callback,
    temperature_task_callback,
    socket_task_callback,
    light_task_callback,
    comm_recv_task_callback,
    comm_sender_task_callback,
    dispatcher_task_callback
};

extern void install_segfault_signal();

static mqd_t maintask_q;

/**
 * @brief  Signal handler for the main task
 *         Should not include stdout log in the handler as it is not
thread safe. Find an alternative for this
 *          Maybe use a global atomic type to set the signal type after
cancelling all the thereads and check that
 *          atomic data in the main_task after the join call if a signal
occured and then use a stdout log there
 *
 * @param signal
 */
static void signal_handler(int signal)
{
      switch (signal)
      {

            case SIGUSR1:
                  LOG_STDOUT(SIGNAL "SIGUSR1 signal.\n");
                  break;
            case SIGUSR2:
                  LOG_STDOUT(SIGNAL "SIGUSR2 signal.\n");
                  break;
            case SIGINT:
                  LOG_STDOUT(SIGNAL "SIGINT signal.\n");
                  break;
            case SIGTERM:
                  LOG_STDOUT(SIGNAL "SIGTERM signal.\n");
                  break;
            case SIGTSTP:
```

```
                LOG_STDOUT(SIGNAL "SIGTSTP signal.\n");
                break;
            default:
                LOG_STDOUT(SIGNAL "Invalid signal.\n");
                break;
      }
    /* Cancelling all the threads for any signals */
    // for(int i = 0; i < NUM_CHILD_THREADS; i++)
    // {
    //     pthread_cancel(pthread_id[i]);
    // }

    signal_exit = 1;
}

/**
 * @brief Timer handler
 *
 * @param sigval
 */
static void timer_handler_setup(union sigval sig)
{
    if(1 == timeoutflag)
    {
        LOG_STDOUT(ERROR "TIMEOUT. App could not be setup in time\n");
        timeoutflag=0;
        delete_timer(*(timer_t*)sig.sival_ptr);
        exit(1);
    }
}

static void timer_handler_aliveStatusCheck(union sigval sig)
{
    if(!aliveStatus[LOGGER_TASK_ID] && !aliveStatus[LIGHT_TASK_ID] &&
!aliveStatus[TEMPERATURE_TASK_ID])
    {
        pthread_mutex_lock(&aliveState_lock);
        aliveStatus[LOGGER_TASK_ID]++;
        aliveStatus[LIGHT_TASK_ID]++;
        aliveStatus[TEMPERATURE_TASK_ID]++;
        pthread_mutex_unlock(&aliveState_lock);
        DEFINE_LOG_STRUCT(logstruct,LT_MSG_TASK_STATUS,MAIN_TASK_ID);

DEFINE_LIGHT_STRUCT(lightstruct,LIGHT_MSG_TASK_STATUS,MAIN_TASK_ID)
        DEFINE_TEMP_STRUCT(tempstruct,TEMP_MSG_TASK_STATUS,MAIN_TASK_ID)
        POST_MESSAGE_LOGTASK(&logstruct,"Send Alive status");
        POST_MESSAGE_LIGHTTASK(&lightstruct);
        POST_MESSAGE_TEMPERATURETASK(&tempstruct);
    }
    else
    {
        LOG_STDOUT(ERROR "One of the task not alive\n");
        stop_timer(*(timer_t*)sig.sival_ptr);
        delete_timer(*(timer_t*)sig.sival_ptr);
        /* Cancelling all the threads for any signals */
        for(int i = 0; i < NUM_CHILD_THREADS && !aliveStatus[i]; i++)
        {
            #ifdef VALUES
```

```
            LOG_STDOUT(INFO "Child thread cancelled: %d %s\n",i,
getTaskIdentfierString(i));
            #endif
            if(pthread_cancel(pthread_id[i]))
                LOG_STDOUT(INFO "Child thread cancelled failed: %d\n",i);
        }
        #ifdef VALUES
        LOG_STDOUT(INFO "All child thread cancelled\n");
        #endif
        BB_LedON(1);
        /* Signaling main task to quit */
        signal_exit = 1;
    }
}

mqd_t getHandle_MainTaskQueue()
{
    return maintask_q;
}

int main_task_init()
{
    struct mq_attr maintaskQ_attr = {
        .mq_msgsize = sizeof(MAINTASKQ_MSG_T),
        .mq_maxmsg = 32,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_MAINTASK_NAME);
    maintask_q = mq_open(MQ_MAINTASK_NAME, O_CREAT | O_RDWR, 0666,
&maintaskQ_attr);

    return maintask_q;;
}

void main_task_processMsg()
{
    int ret,prio;
    MAINTASKQ_MSG_T queueData = {0};
    struct timespec recv_timeout = {0};
    while(!signal_exit)
    {
        memset(&queueData,0,sizeof(queueData));
        clock_gettime(CLOCK_REALTIME, &recv_timeout);
        recv_timeout.tv_sec += 2;
        ret =
mq_timedreceive(maintask_q,(char*)&(queueData),sizeof(queueData),&prio,&r
ecv_timeout);
        if(ERR == ret && ETIMEDOUT == errno)
        {
            continue;
        }
        if(ERR == ret)
        {
            LOG_STDOUT(ERROR "MAIN TASK:MQ_RECV:%s\n",strerror(errno));
            continue;
        }
```

```
        switch(queueData.msgID)
        {
            case(MT_MSG_STATUS_RSP):
                #ifdef STDOUT_ALIVE
                LOG_STDOUT(INFO
"ALIVE:%s\n",getTaskIdentfierString(queueData.sourceID));
                #endif
                pthread_mutex_lock(&aliveState_lock);
                aliveStatus[queueData.sourceID]--;
                pthread_mutex_unlock(&aliveState_lock);
                break;
            default:
                LOG_STDOUT(INFO "Invalid Main task queue id\n");
                break;
        }
    }

    LOG_STDOUT(INFO "MAIN TASK GOT EXIT\n");

}

void POST_EXIT_MESSAGE_ALL()
{
    comm_recv_task_exit = 1;
    POST_MESSAGE_COMM_SENDTASK_EXIT("EXIT");
    POST_MESSAGE_DISPATCHERTASK_EXIT("EXIT");
    LOG_STDOUT(WARNING "FIRE IN THE HOLE. EXIT EXIT!\n");
    DEFINE_LOG_STRUCT(logstruct,LT_MSG_TASK_EXIT,MAIN_TASK_ID);
    DEFINE_LIGHT_STRUCT(lightstruct,LIGHT_MSG_TASK_EXIT,MAIN_TASK_ID)
    DEFINE_TEMP_STRUCT(tempstruct,TEMP_MSG_TASK_EXIT,MAIN_TASK_ID)
    POST_MESSAGE_LIGHTTASK_EXIT(&lightstruct);
    POST_MESSAGE_TEMPERATURETASK_EXIT(&tempstruct);
    pthread_cancel(pthread_id[SOCKET_TASK_ID]);
    POST_MESSAGE_LOGTASK_EXIT(&logstruct,"FIRE IN THE HOLE. EXIT EXIT!");
}

int main_task_entry()
{
    #ifdef SEG_FAULT_HANDLER
    install_segfault_signal();
    #endif
    /* Making the timeout flag true, this should be unset=false within 5
sec else the timer checking the operation
    will send a kill signal and the app will close
    This is to make sure that the barrier is passed within 5 secs. Extra
safety feature which might not be neccessary at all.
    */
    timeoutflag = 1;
    signal_exit = 0;
    int ret = main_task_init();
    if(-1 == ret)
    {
        LOG_STDOUT(ERROR "MAIN TASK INIT:%s\n",strerror(errno));
        return ret;
    }

    ret = configdata_setup();
    if(ret)
```

```
        LOG_STDOUT(ERROR "Could not setup data from config file\n");

    /* Mutex init */
    pthread_mutex_init(&aliveState_lock, NULL);

    /* Registering a timer for 5 sec to check that the barrier is passed
*/
    timer_t timer_id;
    if(ERR == register_and_start_timer(&timer_id, 20*MICROSEC, 1,
timer_handler_setup, &timer_id))
    {
        // LOG_STDOUT(ERROR "Timer Error\n");
        return ERR;
    }

    /* Create a barrier for all the threads + the main task*/
    pthread_barrier_init(&tasks_barrier,NULL,NUM_CHILD_THREADS+1);

    struct sigaction sa;
    /*Registering the signal callback handler*/
      register_signalHandler(&sa,signal_handler, REG_SIG_ALL);

    /* Create all the child threads */
    for(int i = 0; i < NUM_CHILD_THREADS; i++)
    {
        ret =
pthread_create(&pthread_id[i],NULL,thread_callbacks[i],NULL);
        if(ret != 0)
        {
            LOG_STDOUT(ERROR "Pthread create:%d:%s\n",i,strerror(errno));
            return ret;
        }
    }

    LOG_STDOUT(INFO "MAIN TASK INIT COMPLETED\n");
    pthread_barrier_wait(&tasks_barrier);

    /* Resetting the timeoutflag as we are pass the barrier */
    timeoutflag = 0;

    ret= stop_timer(timer_id);
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "MAIN TASK CANNOT STOP
TIMER:%s\n",strerror(errno));
        return ERR;
    }

    ret == delete_timer(timer_id);
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "MAIN TASK CANNOT DELETE
TIMER:%s\n",strerror(errno));
    }

    if(ERR == register_and_start_timer(&timer_id, 5*MICROSEC, 0
,timer_handler_aliveStatusCheck, &timer_id))
    {
```

```
        // LOG_STDOUT(ERROR "Timer Start Error\n");
        return ERR;
    }

    send_GET_CLIENT_INFO_UID(TIVA_BOARD1_ID);
    sleep(1);
    send_GET_CLIENT_INFO_UID(XYZ_TIVA_BOARD_ID);

    /* Start message processing which is a blocking call */
    main_task_processMsg();

    delete_timer(timer_id);

    POST_EXIT_MESSAGE_ALL();

    for(int i = 0; i < NUM_CHILD_THREADS; i++)
    {
        int retThread = 0;
        //  LOG_STDOUT(INFO "Pthread JOIN:%d\n",i);
        ret = pthread_join(pthread_id[i],(void*)&retThread);
        //LOG_STDOUT(INFO "ThreadID %d: Ret:%d\n",i,retThread);
        if(ret  != 0)
        {
            LOG_STDOUT(ERROR "Pthread join:%d:%s\n",i,strerror(errno));
            return ret;
        }
    }

    pthread_mutex_destroy(&aliveState_lock);

    configdata_flush();

    BB_LedOFF(1);

    LOG_STDOUT(INFO "GOODBYE CRUEL WORLD!!!\n");

    return SUCCESS;
}

/**
 * @brief
 *
 * @file my_time.c
 * @author Gunj Manseta
 * @date 2018-03-18
 */
#include <sys/time.h>
#include <time.h>
#include <string.h>

#include "my_time.h"
#include "error_data.h"

#define GET_TIMEOFDAY(x,y)   gettimeofday(x,y)
      //syscall(__sys_gettimeofday,x,y)

int get_time_string(char *timeString, const int len)
{
```

```
      struct timeval tv;
      //struct tm* ptm;
      char time_string[20] = {0};

      /* Obtain the time of day using the system call */
      unsigned long ret = GET_TIMEOFDAY(&tv,NULL);
      if(ret != 0)
      {
            memset(timeString,0,len);
            return ERR;
      }
      snprintf(time_string,sizeof(time_string),"%ld.%ld",tv.tv_sec,tv.tv_
usec);
      //ptm = localtime (&tv.tv_sec);
      /* Format the date and time. */
      //strftime (time_string, sizeof (time_string), "%Y-%m-%d %H:%M:%S",
ptm);
      //strftime (time_string, sizeof (time_string), "%X", ptm);
    memcpy(timeString,time_string, len);

    return SUCCESS;
}
/**
 * @brief
 *
 * @param temp
 * @param unit
 * @return int
 */

#include <string.h>
#include "my_i2c.h"
#include "tmp102_sensor.h"

const TMP102_CONFIG_REG_SETTINGS_T TMP102_CONFIG_DEFAULT =
{
    .SD_MODE = 0,
    .TM_MODE = 0,
    .POL     = 0,
    .OS      = 0,
    .EM_MODE = 0,
    .CR      = 2
};

int TMP102_setmode_allDefault()
{
    int ret = TMP102_setMode_CR_4HZ_default();
    if(ret) return ret;
    ret = TMP102_setMode_SD_Continuous_default();
    if(ret) return ret;
    ret = TMP102_setMode_ALERT_ActiveLow_default();
    if(ret) return ret;
    ret = TMP102_setMode_TM_ComparatorMode_default();
    if(ret) return ret;
    ret = TMP102_setMode_EM_NormalMode_default();
    return ret;
}
int TMP102_setMode(TMP102_CONFIG_REG_SETTINGS_T config)
```

```
{
    int ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,*((uint16_t*)&config), 0);

    return ret;
}

int TMP102_setMode_SD_PowerSaving()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data |= (uint16_t)TMP102_CONFIG_SD;

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_SD_Continuous_default()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_SD);

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;
}

int TMP102_setMode_TM_ComparatorMode_default()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_TM);

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);
```

```
    return ret;

}
int TMP102_setMode_TM_InterruptMode()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data |= (uint16_t)TMP102_CONFIG_TM;

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_ALERT_ActiveLow_default()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_POL);

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_ALERT_ActiveHigh()
{
    uint16_t config_data = 1;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data |= (uint16_t)TMP102_CONFIG_POL;

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_EM_NormalMode_default()
```

```
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_EM);

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;


}
int TMP102_setMode_EM_ExtendedMode()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data |= (uint16_t)TMP102_CONFIG_EM;

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_CR_250mHZ()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_CR(3));

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_CR_1HZ()
{
    uint16_t config_data = 0;
```

```
     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_CR(3));
    config_data |= ((uint16_t)TMP102_CONFIG_CR(1));

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_setMode_CR_4HZ_default()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data &= ~((uint16_t)TMP102_CONFIG_CR(3));
    config_data |= ((uint16_t)TMP102_CONFIG_CR(2));

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}

int TMP102_setMode_CR_8HZ()
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    config_data |= ((uint16_t)TMP102_CONFIG_CR(3));

    ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION, config_data, 0);

    return ret;

}
int TMP102_readMode_ALERT(uint8_t *al_bit)
{
    uint16_t config_data = 0;

     /* Reading the already configured values in the sensor */
```

```
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_CONFIGURATION,(uint8_t*)&config_data, sizeof(config_data));
    if(ret == -1)
        return ret;

    *al_bit = (config_data & ((uint16_t)TMP102_CONFIG_AL))>>13;

    return ret;
}

int TMP102_write_Tlow(float tlow_C)
{
    if(tlow_C < -56.0f || tlow_C > 151.0f)
        tlow_C = 75.0f;

    tlow_C /= 0.0625;
    uint16_t tl;

    if(tlow_C > 0)
    {
        tl = ((uint16_t)tlow_C << 4);
        tl &= 0x7FFF;
    }
    else
    {
        tlow_C = -1 * tlow_C;
        tl = (uint16_t)tlow_C;
        tl = ~(tl) + 1;
        tl = tl << 4;
    }

    int ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR, TMP102_REG_TLOW,
tl, 0);
    if(ret == -1)
        return ret;

    #ifdef TEST_I2C
    uint16_t retTlow =0;
    ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_TLOW,(uint8_t*)&retTlow, sizeof(retTlow));
    if(ret == -1)
        return ret;

    assert(retTlow == tl);
    assert_int_equal(retTlow,tl);
    #endif

}

int TMP102_write_Thigh(float thigh_C)
{

    if(thigh_C < -56.0f || thigh_C > 151.0f)
        thigh_C = 80.0f;

    thigh_C /= 0.0625;
    uint16_t th;
```

```
    if(thigh_C > 0)
    {
        th = ((uint16_t)thigh_C << 4);
        th &= 0x7FFF;
    }
    else
    {
        thigh_C = -1 * thigh_C;
        th = (uint16_t)thigh_C;
        th = ~(th) + 1;
        th = th << 4;
    }

    int ret = I2Cmaster_write_word(TMP102_SLAVE_ADDR, TMP102_REG_TLOW,
th, 0);
    if(ret == -1)
        return ret;

    #ifdef TEST_I2C
    uint16_t ret =0;
    ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_TLOW,(uint8_t*)&ret, sizeof(ret));
    if(ret == -1)
        return ret;

    assert(ret == th);
    assert_true(ret == th);
    #endif


}

int TMP102_read_Tlow(float *tlow_C)
{
    uint16_t tlow =0;

    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_TLOW,(uint8_t*)&tlow, sizeof(ret));
    if(ret == -1)
        return ret;

    if(tlow & 0x800)
    {
        tlow = (~tlow) + 1;
        *tlow_C = (-1) * (float)tlow * 0.0625;
    }
    else
    {
        *tlow_C = ((float)tlow)*0.0625;
    }

    return ret;

}

int TMP102_read_Thigh(float *thigh_C)
{
    uint16_t thigh =0;
```

```
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_TLOW,(uint8_t*)&thigh, sizeof(ret));
    if(ret == -1)
        return ret;

    if(thigh & 0x800)
    {
        thigh = (~thigh) + 1;
        *thigh_C = (-1) * (float)thigh * 0.0625;
    }
    else
    {
        *thigh_C = ((float)thigh)*0.0625;
    }

    return ret;

}


int TMP102_getTemp(float *temp, TEMPERATURE_UNIT_T unit)
{
    uint8_t buff[2] = {0};
    int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR,
TMP102_REG_TEMPERATURE, buff, sizeof(buff));
    if(ret == -1)
        return ret;

    /* We get MSB(15:8) in buff[0] and LSB(7:4) in buff[1] */
    uint16_t temp_raw = 0;
    temp_raw = (((uint16_t)buff[0]) << 4) | (buff[1] >> 4) & 0xFFF;
    if(temp_raw & 0x800)
    {
        temp_raw = ((~temp_raw) + 1) & 0xFFF;
        *temp = (-1) * (float)temp_raw * 0.0625;
    }
    else
    {
        *temp = ((float)temp_raw)*0.0625;
    }

    if(unit == FAHREN)
    {
        *temp = (*temp * 1.8) + 32;
    }
    else if(unit == KELVIN)
    {
        *temp += 273.15;
    }

    return ret;
}

uint16_t* TMP102_memDump()
{
    uint16_t *memdump = (uint16_t*)malloc(4*sizeof(uint16_t));
    if(NULL == memdump)
```

```
        return NULL;

    memset(memdump, 0 , 4);

    for(uint8_t i = 0 ; i < 0x4; i++)
    {
        int ret = I2Cmaster_read_bytes(TMP102_SLAVE_ADDR, i ,
(uint8_t*)(memdump+i), sizeof(uint16_t));
        memdump[i] = (memdump[i]<<8) | (memdump[i]>>8);

        //float temp = (float)(memdump[i]>>4) * 0.0625;
        //printf(" - F:%.02f - ",temp);
    }

    return memdump;
}/**
 * @brief
 *
 * @file my_signals.c
 * @author Gunj Manseta
 * @date 2018-03-18
 */

#include "my_signals.h"
#include "error_data.h"

int register_signalHandler(struct sigaction *sa, void (*handler)(int),
REG_SIGNAL_FLAG_t signalMask)
{
      sa->sa_handler = handler;

      sa->sa_flags = SA_RESTART;

      sigfillset(&sa->sa_mask);

      int ret_error = 0;

      if ((signalMask & REG_SIG_USR1) && sigaction(SIGUSR1, sa, NULL) ==
-1)
      {
            ret_error++;
            LOG_STDOUT(ERROR "Cannot handle SIGUSR1.\n");
      }

      if ((signalMask & REG_SIG_USR2) && sigaction(SIGUSR2, sa, NULL) ==
-1)
      {
            ret_error++;
            LOG_STDOUT(ERROR "Cannot handle SIGUSR2.\n");
      }

      if ((signalMask & REG_SIG_INT) && sigaction(SIGINT, sa, NULL) == -
1)
      {
            ret_error++;
            LOG_STDOUT(ERROR "Cannot handle SIGINT.\n");
      }
```

```
      if ((signalMask & REG_SIG_TSTP) && sigaction(SIGTERM, sa, NULL) ==
-1)
      {
            ret_error++;
            LOG_STDOUT(ERROR "Cannot handle SIGTERM.\n");
      }

      if ((signalMask & REG_SIG_TSTP) && sigaction(SIGTSTP, sa, NULL) ==
-1)
      {
            ret_error++;
            LOG_STDOUT(ERROR "Cannot handle SIGTSTOP.\n");
      }

      return ret_error;
}
/*
 * communication_interface.c
 *
 *  Created on: 22-Apr-2018
 *      Author: Gunj Manseta
 */

#if 1
#include "communication_interface.h"

/* NRF COMM FUNCTIONS*/
void my_NRF_IntHandler()
{
}

volatile uint8_t count = 0;
int8_t comm_init_NRF()
{
    if(count)
    {
        count++;
        return 0;
    }
    int8_t status = NRF_moduleInit(NRF_USE_INTERRUPT, my_NRF_IntHandler);
    if(status == -1)
        return status;
    NRF_moduleSetup(NRF_DR_1Mbps, NRF_PW_MED);
    NRF_openReadPipe(1, RXAddr, sizeof(COMM_MSG_T)>32 ? 32 :
sizeof(COMM_MSG_T));
    NRF_openWritePipe(TXAddr);
    count++;
}

void comm_deinit_NRF()
{
    count--;
    if(count)
    {
        return;
    }
    NRF_closeReadPipe(1);
    NRF_closeWritePipe();
```

```
    NRF_moduleDisable();
}

int32_t comm_sendNRF_raw(uint8_t *data, uint32_t len)
{
    if(len <= 32)
    {
        NRF_transmit_data(data, len, true);
    }
//    else
//    {
//        size_t i = 0;
//        while(i < len)
//        {
//            NRF_transmit_data(data+i, 32 - (i%32), false);
//            i = i+32;
//        }
//    }

}

int32_t comm_recvNRF_raw(uint8_t *data, size_t len)
{

}
int32_t comm_recvNRF(COMM_MSG_T *p_comm_object)
{
    return NRF_read_data((uint8_t*)p_comm_object, sizeof(COMM_MSG_T));
}

#endif
/**
 * @brief
 *
 * @file BB_Led.c
 * @author Gunj Manseta
 * @date 2018-03-10
 */

#include <stdlib.h>
#include <stdio.h>
#include "BB_Led.h"


#define ON "1"
#define OFF "0"


#define LED_COUNT   4
const char *const LEDPATH[LED_COUNT] =
{
    "/sys/class/leds/beaglebone:green:usr0/brightness",
    "/sys/class/leds/beaglebone:green:usr1/brightness",
    "/sys/class/leds/beaglebone:green:usr2/brightness",
    "/sys/class/leds/beaglebone:green:usr3/brightness"
};

int BB_LedON(USER_LED_T lednum)
```

```
{
    /* Forcefully using USR LED 1 */
    lednum = 1;
    if(lednum < 4)
    {
        FILE *led_fd = fopen(LEDPATH[lednum], "r+");
        if(led_fd)
        {
            fwrite(ON,1,1,led_fd);
            fclose(led_fd);
            return 0;
        }
        else
            return -1;
    }
    else
        return -1;

}

int BB_LedOFF(USER_LED_T lednum)
{
    /* Forcefully using USR LED 1 */
    lednum = 1;
    if(lednum < 4)
    {
        FILE *led_fd = fopen(LEDPATH[lednum], "r+");
        if(led_fd)
        {
            fwrite(OFF,1,1,led_fd);
            fclose(led_fd);
            return 0;
        }
        else
            return -1;
    }
    else
        return -1;

}

int BB_LedDefault()
{
    FILE *led_fd = fopen("/sys/class/leds/beaglebone:green:usr0/trigger",
"r+");
    if(led_fd)
    {
        fwrite("heartbeat",1,sizeof("heartbeat"),led_fd);
        fclose(led_fd);
        return 0;
    }
    else
        return -1;
}#include "main_task.h"


int main()
{
```

```
    int ret = main_task_entry();

    return ret;


}/**
 * @brief
 *
 * @file temperature_sensor_task.c
 * @author Gunj Manseta
 * @date 2018-03-11
 */

#include <pthread.h>
#include <fcntl.h>           /* For O_* constants */
#include <sys/stat.h>        /* For mode constants */
#include <mqueue.h>
#include <string.h>
#include <errno.h>

#include "main_task.h"
#include "logger_task.h"
#include "error_data.h"
#include "temperature_sensor_task.h"
#include "my_i2c.h"
#include "tmp102_sensor.h"
#include "common_helper.h"


#define MQ_TEMPERATURETASK_NAME "/temperaturetask_queue"

static mqd_t temperaturetask_q;

pthread_mutex_t tempChangeLock;

volatile static float latest_temperature;


float getTempTask_temperature()
{
    float temp;
    pthread_mutex_lock(&tempChangeLock);
    temp = latest_temperature;
    pthread_mutex_unlock(&tempChangeLock);
    return temp;
}

static void timer_handler_getAndUpdateTemperature(union sigval sig)
{
    float temperature;

    DEFINE_LOG_STRUCT(logtaskstruct,LT_MSG_LOG,TEMPERATURE_TASK_ID);

    int ret = TMP102_getTemp_Celcius(&temperature);
    if(ret == 0)
    {
        #ifdef VALUES
```

```
        LOG_STDOUT(INFO "Celcius:%.03f\n",temperature);
        #endif

        #ifdef LOGVALUES
        POST_MESSAGE_LOGTASK(&logtaskstruct,INFO
"Celcius:%.03f\n",temperature);
        #endif
    }
    else
    {
        LOG_STDOUT(ERROR "Temperature Sensor Inactive\n");
        POST_MESSAGE_LOGTASK(&logtaskstruct,ERROR "Temperature Sensor
Inactive\n");
        return;
    }

    pthread_mutex_lock(&tempChangeLock);
    latest_temperature = temperature;
    pthread_mutex_unlock(&tempChangeLock);
}

mqd_t getHandle_TemperatureTaskQueue()
{
    return temperaturetask_q;
}

/**
 * @brief
 *
 * @return int
 */
int temperature_task_queue_init()
{
    struct mq_attr temperaturetaskQ_attr = {
        .mq_msgsize = sizeof(TEMPERATURETASKQ_MSG_T),
        .mq_maxmsg = 128,
        .mq_flags = 0,
        .mq_curmsgs = 0
    };

    mq_unlink(MQ_TEMPERATURETASK_NAME);
    temperaturetask_q = mq_open(MQ_TEMPERATURETASK_NAME, O_CREAT |
O_RDWR, 0666, &temperaturetaskQ_attr);

    return temperaturetask_q;;
}

void temperature_task_processMsg()
{
    int ret,prio;
    TEMPERATURETASKQ_MSG_T queueData = {0};

DEFINE_MAINTASK_STRUCT(maintaskRsp,MT_MSG_STATUS_RSP,TEMPERATURE_TASK_ID)
;
    DEFINE_LOG_STRUCT(logtaskstruct,LT_MSG_LOG,TEMPERATURE_TASK_ID);
    //struct timespec recv_timeout = {0};
    uint8_t continue_flag = 1;
    while(continue_flag)
```

```
    {
        memset(&queueData,0,sizeof(queueData));
        // clock_gettime(CLOCK_REALTIME, &recv_timeout);
        // recv_timeout.tv_sec += 3;
        // ret =
mq_timedreceive(temperaturetask_q,(char*)&(queueData),sizeof(queueData),&
prio, &recv_timeout);
        ret =
mq_receive(temperaturetask_q,(char*)&(queueData),sizeof(queueData),&prio)
;
        if(ERR == ret)
        {
            LOG_STDOUT(ERROR "MQ_RECV:%s\n",strerror(errno));
            POST_MESSAGE_LOGTASK(&logtaskstruct,ERROR
"MQ_RECV:%s\n",strerror(errno));
            continue;
        }
        switch(queueData.msgID)
        {
            case(TEMP_MSG_TASK_STATUS):
                /* Send back task alive response to main task */
                POST_MESSAGE_LOGTASK(&logtaskstruct,INFO "ALIVE STATUS
by:%s\n",getTaskIdentfierString(queueData.sourceID));
                POST_MESSAGE_MAINTASK(&maintaskRsp, "Temperature sensor
task Alive");
                break;
            case(TEMP_MSG_TASK_GET_TEMP):
                break;
            case(TEMP_MSG_TASK_READ_DATA):
                break;
            case(TEMP_MSG_TASK_WRITE_CMD):
                break;
            case(TEMP_MSG_TASK_POWERDOWN):
                break;
            case(TEMP_MSG_TASK_POWERUP):
                break;
            case(TEMP_MSG_TASK_EXIT):
                continue_flag = 0;
                LOG_STDOUT(INFO "Temperature Task Exit request
from:%s\n",getTaskIdentfierString(queueData.sourceID));
                POST_MESSAGE_LOGTASK(&logtaskstruct,INFO "Temperature
Task Exit request from:%s\n",getTaskIdentfierString(queueData.sourceID));
                break;
            default:
                break;
        }
    }

}

/**
 * @brief
 *
 * @param i2c
 * @return int
 */
int temperature_task_I2Cinit(I2C_MASTER_HANDLE_T *i2c)
{
```

```
    int ret = 0;
    if(ret = I2Cmaster_Init(i2c) !=0)
    {
        printErrorCode(ret);
        LOG_STDOUT(ERROR "[FAIL] I2C Master init failed\n");
    }

    return ret;
}

/**
 * @brief
 *
 * @param i2c
 * @return int
 */
int temperature_task_I2Cdeinit(I2C_MASTER_HANDLE_T *i2c)
{
    int ret = 0;
    ret = I2Cmaster_Destroy(i2c);
    if(ret !=0)
    {
        printErrorCode(ret);
        LOG_STDOUT(WARNING "I2C Master destroy failed\n");
    }

    return ret;
}

void* temperature_task_callback(void *threadparam)
{
    LOG_STDOUT(INFO "TEMPERATURE TASK STARTED\n");

    int ret = temperature_task_queue_init();
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "TEMPERATURE TASK INIT%s\n",strerror(errno));
        exit(ERR);
    }

    I2C_MASTER_HANDLE_T i2c;
    ret = temperature_task_I2Cinit(&i2c);
    if(ret)
    {
        LOG_STDOUT(ERROR "[FAIL] TEMPERATURE TASK SENSOR
INIT:%s\n",strerror(errno));
        goto FAIL_EXIT_SENSOR;
        //exit(ERR);
    }

    if(ret == 0) {LOG_STDOUT(INFO "[OK] Sensor Test\n");}
    else {LOG_STDOUT(INFO "[FAIL] Sensor Test\n");}


    LOG_STDOUT(INFO "TEMPERATURE TASK INIT COMPLETED\n");
    pthread_barrier_wait(&tasks_barrier);
```

```
    /* Registering a timer for 2 sec to update the task temp copy by
getting the temperature value from the sensor*/
    timer_t timer_id;
    if(ERR == register_and_start_timer(&timer_id, 2*MICROSEC, 0,
timer_handler_getAndUpdateTemperature, &timer_id))
    {
        // LOG_STDOUT(ERROR "Timer Error\n");
        goto FAIL_EXIT;
    }

    /* Process Log queue msg which executes untill the log_task_end flag
is set to true*/
    temperature_task_processMsg();


    ret = delete_timer(timer_id);
    if(ERR == ret)
    {
        LOG_STDOUT(ERROR "TEMPERATURE TASK DELETE
TIMER:%s\n",strerror(errno));
        exit(1);
    }

FAIL_EXIT:
    ////* Commented the i2x deint as the light sensor task will deinit
the handle. THe handle within the low level i2c is common for a master */
    temperature_task_I2Cdeinit(&i2c);

FAIL_EXIT_SENSOR:
    mq_close(temperaturetask_q);
    LOG_STDOUT(INFO "Temperature task exit.\n");
    return SUCCESS;
}
```