```
#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "inc/hw_gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"
#include "utils/uartstdio.h"

uint32_t g_ui32SysClock;
int main(void)
{

    float ftest;
    int test;
    //
    // Run from the PLL at 120 MHz.
    //
    g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                SYSCTL_CFG_VCO_480), 120000000);

    //
    // Configure the device pins.
    //
    PinoutSet(false, false);

    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);


    ConfigureUART();

    UARTprintf("UART Configured \n");

    test = ConfigureHumidity();
        if(test == 0)
            UARTprintf("[Success] Sensor Configuring \n");
        else
            UARTprintf("[Failure] Sensor Configuring\n");

   ftest = get_humidity();
   if(ftest > 100 || ftest < 0)
               UARTprintf("[Success] Getting Humidity\n");
           else
               UARTprintf("[Failure] Getting Humidity\n");

   ftest = get_temp();
      if(ftest < 30 || ftest > 10)
                  UARTprintf("[Success] Getting Temperature\n");
          else
                  UARTprintf("[Failure] Getting Temperature\n");
```

```
}
#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "inc/hw_gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"
#include "utils/uartstdio.h"
#include "queue.h>

uint32_t g_ui32SysClock;
int main(void)
{
    int i;
    float ftest;
    int test;
    myqueue = xQueueCreate(5, 10);
    //
    // Run from the PLL at 120 MHz.
    //
    g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                SYSCTL_CFG_VCO_480), 120000000);

    //
    // Configure the device pins.
    //
    PinoutSet(false, false);

    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);


    ConfigureUART();

    UARTprintf("UART Configured \n");

myqueue = xQueueCreate(5, 10);

            if(myqueue == NULL)
                UARTprintf("[Success] Queue Configuring \n");
            else
                UARTprintf("[Failure] Queue Configuring\n");

            char put[10]="Hi there\0";
            char get[10]={0};

            if( xQueueSendToBack( myqueue, &put, strlen(put) ) != pdPASS ){
                 UARTprintf("[Failure] Queue is full \n");
             }

             if( xQueueReceive( myqueue, &get, portMAX_DELAY ) != pdPASS ){
                 UARTprintf("[Failure] Error in queue\n");
```

```
            }

            for(i=0; i<11;i++)
            {
                xQueueSendToBack( myqueue, &put, strlen(put)
            }

            if( xQueueSendToBack( myqueue, &put, strlen(put) ) != pdPASS )
            {
                 UARTprintf("[Failure] Queue is full \n");
            }

            if( xQueueReceive( myqueue, &get, portMAX_DELAY ) != pdPASS ){
                UARTprintf("[Failure] Error in queue receive\n");
            }
            else{
                UARTprintf("[Success] remove successful\n");
            }
            return 0;

}
#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "inc/hw_gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"
#include "utils/uartstdio.h"

uint32_t g_ui32SysClock;
int main(void)
{

    float ftest;
    int test;
        g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                SYSCTL_CFG_VCO_480), 120000000);        PinoutSet(false, false);

    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);


    ConfigureUART();

    UARTprintf("UART Configured \n");

    test = ConfigureAltitude();
        if(test == 0)
            UARTprintf("[Success] Sensor Configuring \n");
        else
            UARTprintf("[Failure] Sensor Configuring\n");
```

```
   ftest = get_altitude();
   if(ftest > 1700 || ftest < 1600)
               UARTprintf("[Success] Getting Altitude\n");
           else
               UARTprintf("[Failure] Getting Altitude\n");

   return 0;

}
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <termios.h>
#include <fcntl.h>
#include <linux/ioctl.h>

struct termios *configure;

int fd;

char *device = "/dev/ttyO4";

/*Function to configure UART*/
void tty_config(struct termios *con, int descriptor)
{
  tcgetattr(descriptor, con);
  con->c_iflag &= ~(IGNBRK | BRKINT | ICRNL | INLCR | PARMRK | INPCK | ISTRIP | IXON);
  con->c_oflag = 0;
  con->c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN | ISIG);
  con->c_cc[VMIN] = 1;
  con->c_cc[VTIME] = 0;
  if(cfsetispeed(con, B9600) || cfsetospeed(con, B9600))
  {
    perror("ERROR in baud set\n");
  }


  if(tcsetattr(descriptor, TCSAFLUSH, con) < 0)
  {
    perror("ERROR in set attr\n");
  }
}

/*Function to initialize UART*/
int uart_init(void)
{

  fd = open(device, O_RDWR | O_NOCTTY | O_SYNC);// | O_NDELAY);
  if(fd == -1)
  {
    perror("ERROR opening file descriptor\n");
  }

  configure = (struct termios*)malloc(sizeof(struct termios));
  tty_config(configure, fd);

  if(tcsetattr(fd,TCSAFLUSH, configure) < 0)
  {
    printf("\nERROR: TC Set Attr\n");
  }
```

```
  return fd;
}

typedef struct packet
{
  uint8_t log_id;
  uint8_t log_level;
  float data;
  char timestamp[25];
  char c;
}log_packet;

void main()
{
  uart_init();

  log_packet rec,recv;

  recv.log_id = 9;
  recv.log_level = 2;
  recv.data = 5.9;

  //read(fd,&rec,sizeof(rec));

  //read(fd,&rec,sizeof(rec));
  char x='a';
  volatile char y=5;

  int n;
printf("%d\n",fd );
printf("Sending Data\n");
  if((n=write(fd,&x,sizeof(x))) < 0){
        printf("\nWrite Fail\n");
}


//printf("Waiting for recv data\n");
if((n=read(fd,&y,sizeof(y))) < 0)
{
    printf("\nRead Fail\n");
}

if( y == x )
  printf("[Success] UART Loopback \n");
else
  printf("[Failure] UART Loopback \n");

}#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>
//#include "main_task.h"


extern float alti,humid;

#define PORT_ADR    2000
```

```
typedef struct
{
  char    buf[20];
  int  buf_len;
  bool    usrLED_OnOff;
}payload_t;


int socket_task()
{
  struct sockaddr_in addr, peer_addr;
  int addr_len = sizeof(peer_addr);
  char rdbuff[1024] = {0};
  int server_socket, accepted_soc, opt = 1;
  int i = 0;
  payload_t *ploadptr;
  int read_b;
  int pload_len = 0;
  char ackbuf[50];
  float temp,lumen;


  /* create socket */
  if((server_socket = socket(AF_INET,SOCK_STREAM,0)) == 0)
  {
    printf("[Server] [ERROR] Socket Creation Error\n");
    return 1;
  }
  else
    printf("[Server] Socket Created Successfully\n");

  /* set socket options */
  if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &(opt), sizeof(opt)))
  {
    printf("[Server] [ERROR] Socket options set error\n");
    return 1;
  }

  /*Set the sockaddr_in structure */
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(PORT_ADR);

  /*bind socket to a address */
  if((bind(server_socket,(struct sockaddr*)&addr, sizeof(addr))) < 0)
  {
    printf("[Server] [ERROR] Bind socket Error\n");
    return 1;
  }
  else
    printf("[Server] Socket binded Successfully\n");



  /* listen for connections*/
  if(listen(server_socket,5) < 0)
  {
    printf("[Server] [ERROR] Can't listen connection\n");
    return 1;
  }
while(1)
 {
```

```
  /*accept connection */
  accepted_soc = accept(server_socket, (struct sockaddr*)&peer_addr,
(socklen_t*)&addr_len);
  if(accepted_soc < 0)
  {
    printf("[Server] [ERROR] Can't accept connection\n");
    return 1;
  }

  /* read payload length */
  read_b = read(accepted_soc, &pload_len, sizeof(int));
  if(read_b == sizeof(int))
  {
    printf("[Server] Size of incoming payload: %d\n",pload_len);
  }
  else
  {
    printf("[Server] [ERROR] Invalid data\n");
    return 1;
  }

  /* read payload */
  while((read_b = read(accepted_soc, rdbuff+i, 1024)) < pload_len)
  {
    i+=read_b;
  }
  ploadptr= (payload_t*)rdbuff;
  /* display data */
  printf("[Server] Message Recvd from Client\n{\n Message:%s\n MessageLen:%d\n USRLED:
%d\n}\n",ploadptr->buf, ploadptr->buf_len, ploadptr->usrLED_OnOff);

  // printf("ID = %d\n", rec.log_id);
  // printf("Data = %f\n", rec.data);

  if(strcmp(ploadptr->buf,"get_humidity")==0)
  {
    snprintf(ackbuf, 50, "Humidity = %f",humid);
    send(accepted_soc , ackbuf , 50, 0);
  }
  else if(strcmp(ploadptr->buf,"get_altitude")==0)
  {
        snprintf(ackbuf, 50, " Altitude = %f",alti);
    send(accepted_soc , ackbuf , 50, 0);
  }
  else
  {
        printf("I Don't Understand !!");
    send(accepted_soc , "I Don't Understand !!" , 50, 0);
  }

  /* send message from server to client */
//  send(accepted_soc , "ACK" , 4, 0);
//  printf("[Server] Message sent from Server: ACK\n");

  /*close socket */
  close(accepted_soc);
        }
  return 0;
}
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
```

```
#include <string.h>
#include <termios.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include "comm_task.h"

struct termios *configure;

int fd;

char *device = "/dev/ttyO4";

char *device_spi = "/dev/spidev1.0";

typedef struct packet
{
  uint8_t log_id;
  uint8_t log_level;
  float data;
  char timestamp[25];
  char c;
}log_packet;



/*Function to configure UART*/
void tty_config(struct termios *con, int descriptor)
{
  tcgetattr(descriptor, con);
  con->c_iflag &= ~(IGNBRK | BRKINT | ICRNL | INLCR | PARMRK | INPCK | ISTRIP | IXON);
  con->c_oflag = 0;
  con->c_lflag &= ~(ECHO | ECHONL | ICANON | IEXTEN | ISIG);
  con->c_cc[VMIN] = 1;
  con->c_cc[VTIME] = 0;
  if(cfsetispeed(con, B9600) || cfsetospeed(con, B9600))
  {
    perror("ERROR in baud set\n");
  }


  if(tcsetattr(descriptor, TCSAFLUSH, con) < 0)
  {
    perror("ERROR in set attr\n");
  }
}

/*Function to initialize UART*/
int uart_init(void)
{

  fd = open(device, O_RDWR | O_NOCTTY | O_SYNC);// | O_NDELAY);
  if(fd == -1)
  {
    perror("ERROR opening file descriptor\n");
  }

  configure = (struct termios*)malloc(sizeof(struct termios));
  tty_config(configure, fd);

  if(tcsetattr(fd,TCSAFLUSH, configure) < 0)
  {
    printf("\nERROR: TC Set Attr\n");
```

```
  }

  return fd;
}

int spi_init(void)
{

  fd = open(device_spi, O_RDWR | O_NOCTTY | O_SYNC);// | O_NDELAY);
  if(fd == -1)
  {
    perror("ERROR opening file descriptor\n");
  }



  return fd;
}
#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>


#define PORT_ADR    2000
#define IP_ADR      "127.0.0.1" /* Loppback IP Address*/

typedef struct
{
  char    buf[20];
  size_t  buf_len;
  bool    usrLED_OnOff;
}payload_t;


int main()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  char msg[20] = "Message from Client";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;*/

while(1)
{
  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
```

```
  else
    printf("[Client] Socket Created Successfully\n");

  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(PORT_ADR);




  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }



  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }


    printf("\n\n Enter The API Message :");
  gets(&msg);



  printf("\n You entered : %s\n",msg);
  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n
USRLED: %d\n}\n", \
                           ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  read(client_socket, r_data, 25);
  printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
```

```
  close(client_socket);
  }
  return 0;
}
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include <unistd.h>



int fd;

char *device = "/dev/spidev1.0";


/*Function to initialize SPI*/
int spi_init(void)
{

  fd = open(device, O_RDWR | O_NOCTTY | O_SYNC);// | O_NDELAY);
  if(fd == -1)
  {
    perror("ERROR opening file descriptor\n");
  }



  return fd;
}

typedef struct packet
{
  uint8_t log_id;
  uint8_t log_level;
  float data;
  char timestamp[50];
  char c;
}log_packet;

void main()
{
  spi_init();

  log_packet rec,recv;

  recv.log_id = 9;
  recv.log_level = 2;
  recv.data = 5.9;

  //read(fd,&rec,sizeof(rec));

  //read(fd,&rec,sizeof(rec));
  char x='a';
  volatile char y=5;

  int n;
printf("%d\n",fd );
// printf("Sending Data\n");
```

```
//   if((n=write(fd,&x,sizeof(x))) < 0){
//         printf("\nWrite Fail\n");
// }

while(1){
//printf("Waiting for recv data\n");
   if((n=read(fd,&rec,sizeof(rec))) < 0)
   {
         printf("\nRead Fail\n");
         }
else if(n>0)
{
  //printf("y= %c\n", y);
  printf("Log ID = %d\n",rec.log_id );
  printf("Log Level = %d\n",rec.log_level );
  printf("Data = %f\n\n",rec.data );
  printf("Data recv\n");

         if(rec.log_id == 1)
                 printf("Altitude = %f\n",rec.data);

         if(rec.log_id == 2)
                 printf("Humidity = %f\n\n",rec.data);
}
}
}#include "main_task.h"
#include <mqueue.h>
#include <float.h>
#include <signal.h>

#define UART_COMM
//#define SPI_COMM

#define HB_PORT_ADR 5000
#define IP_ADR      "127.0.0.1"

pthread_t comm_id, logger_id, alert_id, socket_id;

char* PATH = "/sys/class/leds/beaglebone:green:usr1/trigger";
char* LEDPATH = "/sys/class/leds/beaglebone:green:usr1/brightness";

void remove_trigger(void) {
        FILE* fp = NULL;
        if((fp = fopen(PATH, "r+")))
        {
                fwrite("none", 1, 4, fp);
                fclose(fp);
        }
        else
                printf("Error\n");
}


void LEDOff(void)
{
        FILE* LED = NULL;
        remove_trigger();
        if((LED = fopen(LEDPATH, "r+")))
        {
                fwrite("0", 1, 1, LED);
                fclose(LED);
        }
```

```
        else
                printf("LEDOff error\n");
}


void LEDOn(void)
{
        FILE* LED = NULL;
        remove_trigger();
        if((LED = fopen(LEDPATH, "r+")))
        {
                fwrite("1", 1, 1, LED);
                fclose(LED);
        }
        else
                printf("LEDOn error\n");
}


void kill_all()
{


  printf("<<<Killing all threads>>>\n");
  mqd_t my_queue;
  log_packet my_msg;
  my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
  my_msg.log_level = 0;
  my_msg.log_id = 7;
  mq_send(my_queue,(char *)&my_msg,sizeof(my_msg),1);

  pthread_cancel(comm_id);
  pthread_cancel(alert_id);
  pthread_cancel(socket_id);
  pthread_cancel(logger_id);

}

int comm_client()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  const char* msg = "Comm Task Alive";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;

  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    //printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
  else
    //printf("[Client] Socket Created Successfully\n");
```

```
  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(HB_PORT_ADR);

  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    //printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }

  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    //printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  //printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    //printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  //printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n
USRLED: %d\n}\n", \
                            ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  //read(client_socket, r_data, 4);
  //printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
  close(client_socket);

  //return 0;

}

void *func_comm()
{
        int i;
        int n,fd,fd_spi;
        mqd_t mq1;
  log_packet my_msg;


  #ifdef UART_COMM

    fd = uart_init();

  #endif
```

```
  #ifdef SPI_COMM

    fd = spi_init();

  #endif


        printf("[Communication Thread] Communication Thread Started\n");
             mq1 = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);

         my_msg.log_id = 5;
         my_msg.log_level = 0;
         mq_send(mq1,(char *)&my_msg,sizeof(rec),1);

        while(1)
        {
                // printf("Signal from Comm Task\n");
                // for(i=0;i<1000000000;i++);
                if((n=read(fd,&rec,sizeof(rec))) < 0)
            {
                 printf("\n[UART] Read Fail\n");
                }
                else if(n>0)
                {
                        //printf("y= %c\n", y);
                        //printf("Log ID = %d  ",rec.log_id );
                        //printf("%d\n",rec.log_level );
                        //printf("Data = %f\n\n",rec.data );
                        //printf("Data recv\n");
                        if(rec.log_level == 1)
                        {
                                if(rec.log_id == 1)
                                {
                                        alti=rec.data;
                                        printf("Altitude = %f\n",rec.data);
                                        //printf("Alti = %f\n",alti);
                                        mq_send(mq1,(char *)&rec,sizeof(rec),1);
                                }

                                if(rec.log_id == 2)
                                {
                                        humid=rec.data;
                                        printf("Humidity = %f\n",rec.data);
                                        //printf("Humid = %f\n\n",humid);
                                        mq_send(mq1,(char *)&rec,sizeof(rec),1);
                                }
      }
      else if(rec.log_level == 2)
      {
        if(rec.log_id == 1)
                                {

                                        printf("[Altitude Task] Error in altitude task
\n");
                                        mq_send(mq1,(char *)&rec,sizeof(rec),1);
          kill_all();
                                }

                                if(rec.log_id == 2)
                                {
                                        printf("[Humidity Task] Error in humidity task
\n");
```

```
                                        mq_send(mq1,(char *)&rec,sizeof(rec),1);
                                }
            }
                }
                comm_client();

         }

         printf("[Communication Thread] Communication Thread Finished\n");

}

float get_altitude()
{
        printf("Alti req = %f\n",alti);
        return alti;
}

void *func_logger()
{
        printf("[Logger Thread] Logger Thread Started\n");
    FILE *fptr;
    mqd_t my_queue;
        log_packet given;
    fptr = fopen("log.txt","w");   //use logger_thread -> filename
        fprintf(fptr,"In logger task of BBB.\n");
    my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
    struct mq_attr *pact;
    pact = malloc(sizeof(struct mq_attr));
    mq_getattr(my_queue,pact);
    //fprintf(fptr,"Message queue initialised\n");
    printf("[Logger Thread] Message queue initialised.\n");
    fclose(fptr);
        while(1)
        {
        fptr = fopen("log.txt","a");
                mq_receive(my_queue,(char *)&given,pact->mq_msgsize,NULL);
                if(given.log_id == 1)
                {
           if(given.log_level == 1)
           {
                 fprintf(fptr,"Timestamp:%s, Log level:%d, Log ID:%d, Altitude is: %f
\n",given.timestamp,given.log_level,given.log_id,given.data);
            }
            else if(given.log_level == 2)
            {
              fprintf(fptr,"Error in Altitude thread, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
            }
                }
                else if (given.log_id == 2)
                {
                 if(given.log_level == 1)
           {
                 fprintf(fptr,"Timestamp:%s, Log level:%d, Log ID:%d, Humidity is: %f
\n",given.timestamp,given.log_level,given.log_id,given.data);
           }
            else if(given.log_level == 2)
            {
              fprintf(fptr,"Error in umidity thread, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
            }
```

```
                }
    else if (given.log_id == 4)
    {
           if(given.log_level == 0)
           {
           fprintf(fptr,"Startup test successfull, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
            }
            else if(given.log_level == 2)
            {
               fprintf(fptr,"Startup test failed, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
            }
    }
    else if (given.log_id == 5)
    {

           fprintf(fptr,"Communication thread started, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
    }

    else if (given.log_id == 6)
    {

           fprintf(fptr,"Alert thread started, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
    }
        else if (given.log_id == 7)
    {

           fprintf(fptr,"Gracefully exited, Log level:%d, Log ID:%d
\n",given.log_level,given.log_id);
    }
        fclose(fptr);
        }
    printf("[Logger Thread] Terminating message queue\n");
    printf("[Logger Thread] Logger Thread Finished\n");
        return fptr;
}

void *func_socket()
{
        printf("[Socket Thread] Socket Thread Started\n");
        socket_task();
        printf("[Socket Thread] Socket Thread Finished\n");
}

int alert_client()
{
  int client_socket = 0;
  struct sockaddr_in serv_addr = {0};
  const char* msg = "Alert Task Alive";
  payload_t ploadSend;
  int sent_b;
  size_t pload_size;
  char r_data[4] = {0};

  /* Enter the message into payload structure */
  memcpy(ploadSend.buf,msg,strlen(msg)+1);
  ploadSend.buf_len = strlen(ploadSend.buf);
  ploadSend.usrLED_OnOff = 1;
```

```
  /* create socket */
  if ((client_socket = socket(AF_INET, SOCK_STREAM, 0)) < 0)
  {
    //printf("[Client] [ERROR] Socket creation Error\n");
    return -1;
  }
  else
    //printf("[Client] Socket Created Successfully\n");

  /* Fill the socket address structure */
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_port = htons(HB_PORT_ADR);

  /* convert the IP ADDR to proper format */
  if(inet_pton(AF_INET, IP_ADR, &serv_addr.sin_addr)<=0)
  {
    //printf("[Client] [ERROR] Address Conversion Error\n");
    return -1;
  }

  /* connect the socket before sending the data */
  if (connect(client_socket, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
  {
    //printf("[Client] [ERROR] Connection Failed \n");
    return -1;
  }

  /*send the size of the incoming payload */
  pload_size = sizeof(ploadSend);
  sent_b = send(client_socket,&pload_size,sizeof(size_t), 0);
  //printf("[Client] Sent payload size: %d\n", pload_size);

  /*Sending the payload */
  sent_b = send(client_socket , (char*)&ploadSend , sizeof(ploadSend), 0 );
  /* check whether all the bytes are sent or not */
  if(sent_b < sizeof(ploadSend))
  {
    //printf("[Client] [ERROR] Complete data not sent\n");
    return 1;
  }

  /* display the date sent */
  //printf("[Client] Message sent from Client\n{\n Message: %s\n MessageLen: %d\n
USRLED: %d\n}\n", \
                           ploadSend.buf, ploadSend.buf_len, ploadSend.usrLED_OnOff);

  /* read data sent by server */
  //read(client_socket, r_data, 4);
  //printf("[Client]  Message received from Server: %s\n",r_data);

  /* close socket */
  close(client_socket);

  //return 0;

}

void *func_alert()
{
        int alti_flag=0,humid_flag=0,p;
        printf("[Alert Thread] Alert Thread Started\n");
        mqd_t my_queue;
```

```
  log_packet my_msg;
  my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
  my_msg.log_level = 0;
  my_msg.log_id = 6;
  mq_send(my_queue,(char *)&my_msg,sizeof(my_msg),1);

        while(1)
        {
                if(alti >= 8843)
                {
                        //printf("[Alert Task]You have reached top of the world!!!
\n");
                        alti_flag = 1;
      LEDOn();
                }
                else
                {
                        alti_flag = 0;
      LEDOff();
                }

                if(humid >= 20)
                {
                        //printf("[Alert Task]Humidity less than 5!!!\n");
                        humid_flag = 1;
                        LEDOn();
                }
                else
                {
                        humid_flag = 0;
                        LEDOff();
                }

                alert_client();

                for(p=0;p<50000000;p++);
        }

        printf("[Alert Thread] Alert Thread Finished\n");
}

int check_status()
{
  struct sockaddr_in addr, peer_addr;
  int addr_len = sizeof(peer_addr);
  char rdbuff[1024] = {0};
  int server_socket, accepted_soc, opt = 1;
  int i = 0;
  payload_t *ploadptr;
  int read_b;
  size_t pload_len = 0;

  /* create socket */
  if((server_socket = socket(AF_INET,SOCK_STREAM,0)) == 0)
  {
    printf("[HBServer] [ERROR] Socket Creation Error\n");
    return 1;
  }
  else
    printf("[HBServer] Socket Created Successfully\n");

  /* set socket options */
```

```
  if (setsockopt(server_socket, SOL_SOCKET, SO_REUSEADDR, &(opt), sizeof(opt)))
  {
    printf("[HBServer] [ERROR] Socket options set error\n");
    return 1;
  }

  /*Set the sockaddr_in structure */
  addr.sin_family = AF_INET;
  addr.sin_addr.s_addr = INADDR_ANY;
  addr.sin_port = htons(HB_PORT_ADR);

  /*bind socket to a address */
  if((bind(server_socket,(struct sockaddr*)&addr, sizeof(addr))) < 0)
  {
    printf("[HBServer] [ERROR] Bind socket Error\n");
    return 1;
  }
  else
    printf("[HBServer] Socket binded Successfully\n");

  /* listen for connections*/
  if(listen(server_socket,5) < 0)
  {
    printf("[HBServer] [ERROR] Can't listen connection\n");
    return 1;
  }
while(1)
{
  /*accept connection */
  accepted_soc = accept(server_socket, (struct sockaddr*)&peer_addr,
(socklen_t*)&addr_len);
  if(accepted_soc < 0)
  {
    printf("[HBServer] [ERROR] Can't accept connection\n");
    return 1;
  }

  // read payload length
  read_b = read(accepted_soc, &pload_len, sizeof(size_t));
  if(read_b == sizeof(size_t))
  {
    //printf("[HBServer] Size of incoming payload: %d\n",pload_len);
  }
  else
  {
    //printf("[HBServer] [ERROR] Invalid data\n");
    return 1;
  }

  // read payload
  while((read_b = read(accepted_soc, rdbuff+i, 1024)) < pload_len)
  {
    i+=read_b;
  }
  ploadptr= (payload_t*)rdbuff;
  /* display data */
  printf("[HBServer]  Message: %s\n",ploadptr->buf);

  // send message from server to client
  //send(accepted_soc , "ACK" , 4, 0);
  //printf("[HBServer] Message sent from Server: ACK\n");
}
```

```
  /*close socket */
  close(accepted_soc);

  return 0;
}

int startup_test()
{
        int x=1;

        if(pthread_create(&comm_id, NULL, func_comm, NULL) != 0)
        {
                x=0;
        }

        if(pthread_create(&alert_id, NULL, func_alert, NULL) != 0)
        {
                x=0;
        }

        if(pthread_create(&socket_id, NULL, func_socket, NULL) != 0)
        {
                x=0;
        }

        return x;
}

int main()
{
  mqd_t my_queue;
        int startup_check = startup_test();

        pthread_create(&logger_id, NULL, func_logger, NULL);
  log_packet my_msg;
        my_queue = mq_open("/my_queue",O_RDWR | O_CREAT, 0666, NULL);
        if(startup_check == 1)
        {
                printf("Startup Success!!\n\n");
    my_msg.log_id= 4;
    my_msg.log_level = 0;
    mq_send(my_queue,(char *)&my_msg,sizeof(my_msg),1);



        }
        else if(startup_check == 0)
        {
                printf("\n<<<Startup Test Failed>>>\n\n");
                printf("[Main Task] Killing All Tasks\n");
                pthread_cancel(logger_id);
                pthread_cancel(comm_id);
                pthread_cancel(alert_id);
    my_msg.log_id= 4;
    my_msg.log_level = 2;
    mq_send(my_queue,(char *)&my_msg,sizeof(my_msg),1);
        }

        check_status();

        pthread_join(logger_id,NULL);
        pthread_join(comm_id,NULL);
```

```
        pthread_join(alert_id,NULL);
        pthread_exit(NULL);

        printf("Main Process Terminated\n");
        return 0;
}
        #include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <termios.h>
#include <fcntl.h>
#include <linux/ioctl.h>

void tty_config(struct termios *con, int descriptor);

int uart_init(void);

int spi_init(void);#include <string.h>
#include <stdio.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdbool.h>


int socket_task();#include <stdio.h>
#include <pthread.h>
#include <linux/ioctl.h>
#include "comm_task.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <termios.h>
#include <fcntl.h>
#include <linux/ioctl.h>
#include "socket_task.h"


typedef struct packet
{
  uint8_t log_id;
  uint8_t log_level;
  float data;
  char timestamp[25];
  char c;
}log_packet;

typedef struct
{
  char    buf[20];
  int  buf_len;
  bool    usrLED_OnOff;
}payload_t;

log_packet rec;

float alti=0,humid=0;
```

```
float get_altitude();

void *func_comm();

void *func_logger();

void *func_alert();

int startup_test();

int main();
/*
 * main.h
 *
 *  Created on: Mar 28, 2015
 *      Author: akobyljanec
 */

#ifndef MAIN_H_
#define MAIN_H_

// System clock rate, 120 MHz
#define SYSTEM_CLOCK    120000000U


#endif /* MAIN_H_ */
/*
 * humiditysensor.c
 *
 *  Created on: Apr 28, 2018
 *      Author: Anay
 */
#include <stdarg.h>
#include <stdbool.h>
#include <math.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "humiditysensor.h"


int ConfigureHumidity(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C0);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    GPIOPinConfigure(GPIO_PB2_I2C0SCL);
    GPIOPinConfigure(GPIO_PB3_I2C0SDA);
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_2);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_3);

    I2CMasterInitExpClk(I2C0_BASE, 120000000U, false);
    return 0;
}
```

```
float get_humidity(void)
{

    uint8_t msb, lsb;
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, false);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
    I2CMasterDataPut(I2C0_BASE, 0xE5);

    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);

    while(I2CMasterBusy(I2C0_BASE));
    msb = I2CMasterDataGet(I2C0_BASE);
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);

    while(I2CMasterBusy(I2C0_BASE));
    lsb = I2CMasterDataGet(I2C0_BASE);

    float humidity = (((msb * 256 + lsb) * 125.0) / 65536.0) - 6;
    return humidity;
}


float get_temp(void){

    uint8_t msb, lsb;
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, false);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_SINGLE_SEND);
    I2CMasterDataPut(I2C0_BASE, 0xE3);

    while(I2CMasterBusy(I2C0_BASE));
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_START);

    while(I2CMasterBusy(I2C0_BASE));
    msb = I2CMasterDataGet(I2C0_BASE);
    I2CMasterSlaveAddrSet(I2C0_BASE, 0x40, true);
    I2CMasterControl(I2C0_BASE, I2C_MASTER_CMD_BURST_RECEIVE_FINISH);

    while(I2CMasterBusy(I2C0_BASE));
    lsb = I2CMasterDataGet(I2C0_BASE);

    float temp = (((msb * 256 + lsb) * 175.72) / 65536.0) - 46.85;

    return temp;
}

/*
 * altitudesensor.h
 *
 *  Created on: Apr 28, 2018
 *      Author: Anay Gondhalekar
 */

#ifndef DRIVERS_ALTITUDESENSOR_H_
#define DRIVERS_ALTITUDESENSOR_H_


void i2c_read(char addre, unsigned long data,short *receive);
```

```
unsigned long i2c_write(char address, unsigned long reg,unsigned long data);
int ConfigureAltitude(void);
float get_altitude(void);

#endif /* DRIVERS_ALTITUDESENSOR_H_ */
/*
 * humiditysensor.h
 *
 *  Created on: Apr 28, 2018
 *      Author: Anay
 */

#ifndef DRIVERS_HUMIDITYSENSOR_H_
#define DRIVERS_HUMIDITYSENSOR_H_

int ConfigureHumidity(void);
float get_humidity(void);
float get_temp(void);

#endif /* DRIVERS_HUMIDITYSENSOR_H_ */
//*****************************************************************************
//
// pinout.h - Prototype for the function to configure the device pins on the
//            EK-TM4C1294XL.
//
// Copyright (c) 2013-2014 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 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
//
//*****************************************************************************

#ifndef __DRIVERS_PINOUT_H__
#define __DRIVERS_PINOUT_H__

//*****************************************************************************
//
// If building with a C++ compiler, make all of the definitions in this header
// have a C binding.
//
//*****************************************************************************
#ifdef __cplusplus
extern "C"
{
#endif

//*****************************************************************************
//
// Define Board LED's
```

```
//
//*****************************************************************************
#define CLP_D1              1
#define CLP_D2              2
#define CLP_D3              4
#define CLP_D4              8

#define CLP_D1_PORT         GPIO_PORTN_BASE
#define CLP_D1_PIN          GPIO_PIN_1

#define CLP_D2_PORT         GPIO_PORTN_BASE
#define CLP_D2_PIN          GPIO_PIN_0

#define CLP_D3_PORT         GPIO_PORTF_BASE
#define CLP_D3_PIN          GPIO_PIN_4

#define CLP_D4_PORT         GPIO_PORTF_BASE
#define CLP_D4_PIN          GPIO_PIN_0

//*****************************************************************************
//
// Prototypes.
//
//*****************************************************************************
extern void PinoutSet(bool bEthernet, bool bUSB);
extern void LEDWrite(uint32_t ui32LEDMask, uint32_t ui32LEDValue);
extern void LEDRead(uint32_t *pui32LEDValue);

//*****************************************************************************
//
// Mark the end of the C bindings section for C++ compilers.
//
//*****************************************************************************
#ifdef __cplusplus
}
#endif

#endif // __DRIVERS_PINOUT_H__
/*
 * altitudesensor.c
 *
 *  Created on: Apr 28, 2018
 *      Author: Anay Gondhalekar
 */
#include <stdarg.h>
#include <stdbool.h>
#include <math.h>
#include <stdint.h>
#include "inc/hw_i2c.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_gpio.h"
#include "driverlib/i2c.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "altitudesensor.h"

void i2c_read(char addre, unsigned long data,short *receive)
{
    unsigned long k=0;
    I2CMasterSlaveAddrSet(I2C5_BASE,addre,false); //false
```

```
    I2CMasterDataPut(I2C5_BASE,data);
    I2CMasterControl(I2C5_BASE,I2C_MASTER_CMD_BURST_SEND_START);
    while(!I2CMasterBusy(I2C5_BASE));
    //SysCtlDelay(10);
    while(I2CMasterBusy(I2C5_BASE));
    I2CMasterSlaveAddrSet(I2C5_BASE,addre,true);

    I2CMasterControl(I2C5_BASE,I2C_MASTER_CMD_SINGLE_RECEIVE);

    while(!I2CMasterBusy(I2C5_BASE));

    while(I2CMasterBusy(I2C5_BASE));

    k=I2CMasterErr(I2C5_BASE);

    k=I2CMasterDataGet(I2C5_BASE);

    *receive=k;
    I2CMasterControl(I2C5_BASE,I2C_MASTER_CMD_BURST_RECEIVE_FINISH);


}

unsigned long i2c_write(char address, unsigned long reg,unsigned long data)
{
    unsigned long k=0;
    I2CMasterSlaveAddrSet(I2C5_BASE,address,false);
    I2CMasterDataPut(I2C5_BASE,reg);
    I2CMasterControl(I2C5_BASE,I2C_MASTER_CMD_BURST_SEND_START);
        while(!I2CMasterBusy(I2C5_BASE));

    while(I2CMasterBusy(I2C5_BASE));
    k=I2CMasterErr(I2C5_BASE);
    I2CMasterDataPut(I2C5_BASE,data);
    k=I2CMasterErr(I2C5_BASE);
    I2CMasterControl(I2C5_BASE,I2C_MASTER_CMD_BURST_SEND_FINISH);
    k=I2CMasterErr(I2C5_BASE);


return k;
}

int ConfigureAltitude(void)
{
    short k = 0;
    SysCtlPeripheralEnable(SYSCTL_PERIPH_I2C5);
    SysCtlPeripheralReset(SYSCTL_PERIPH_I2C5);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

    GPIOPinConfigure(GPIO_PB4_I2C5SCL);
    GPIOPinConfigure(GPIO_PB5_I2C5SDA);
    GPIOPinTypeI2CSCL(GPIO_PORTB_BASE, GPIO_PIN_4);
    GPIOPinTypeI2C(GPIO_PORTB_BASE, GPIO_PIN_5);

    I2CMasterInitExpClk(I2C5_BASE, 120000000U, false);
    //SysCtlDelay(g_ui32SysClock / 2 / 3);
    i2c_read(0x60,0x0C, &k);
    k=i2c_write(0x60, 0x26,0xBB); //for one by one send
    k=i2c_write(0x60, 0x27,0x02);
    i2c_read(0x60,0x26, &k);
    i2c_read(0x60,0x27, &k);
    return 0;
```

```
}

float get_altitude(void)
{
    short  data[4]={0,0,0,0};
    float convert = 0;

    i2c_read(0x60,0x01, &data[0]);
    i2c_read(0x60,0x02, &data[1]);
    i2c_read(0x60,0x03, &data[2]);
    convert=(float)((short)(data[0]<<8)|(data[1])) + (float)(data[2]>>4)*0.0625 - 70;
    return convert;
}


//*****************************************************************************
//
// pinout.c - Function to configure the device pins on the EK-TM4C1294XL.
//
// Copyright (c) 2013-2014 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 2.1.0.12573 of the EK-TM4C1294XL Firmware Package.
//
//*****************************************************************************

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_gpio.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "drivers/pinout.h"

//*****************************************************************************
//
//! \addtogroup pinout_api
//! @{
//
//*****************************************************************************

//*****************************************************************************
//
//! Configures the device pins for the standard usages on the EK-TM4C1294XL.
//!
```

```
//! \param bEthernet is a boolean used to determine function of Ethernet pins.
//! If true Ethernet pins are  configured as Ethernet LEDs.  If false GPIO are
//! available for application use.
//! \param bUSB is a boolean used to determine function of USB pins. If true USB
//! pins are configured for USB use.  If false then USB pins are available for
//! application use as GPIO.
//!
//! This function enables the GPIO modules and configures the device pins for
//! the default, standard usages on the EK-TM4C1294XL.  Applications that
//! require alternate configurations of the device pins can either not call
//! this function and take full responsibility for configuring all the device
//! pins, or can reconfigure the required device pins after calling this
//! function.
//!
//! \return None.
//
//*****************************************************************************
void
PinoutSet(bool bEthernet, bool bUSB)
{
    //
    // Enable all the GPIO peripherals.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOG);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOH);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOJ);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOK);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOL);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOM);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPION);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOP);
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOQ);

    //
    // PA0-1 are used for UART0.
    //
    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    //
    // PB0-1/PD6/PL6-7 are used for USB.
    // PQ4 can be used as a power fault detect on this board but it is not
    // the hardware peripheral power fault input pin.
    //
    if (bUSB)
    {
//        HWREG(GPIO_PORTD_BASE + GPIO_O_LOCK) = GPIO_LOCK_KEY;
//        HWREG(GPIO_PORTD_BASE + GPIO_O_CR) = 0xff;
//        ROM_GPIOPinConfigure(GPIO_PD6_USB0EPEN);
        ROM_GPIOPinTypeUSBAnalog(GPIO_PORTB_BASE, GPIO_PIN_0 | GPIO_PIN_1);
//        ROM_GPIOPinTypeUSBDigital(GPIO_PORTD_BASE, GPIO_PIN_6);
        ROM_GPIOPinTypeUSBAnalog(GPIO_PORTL_BASE, GPIO_PIN_6 | GPIO_PIN_7);
        ROM_GPIOPinTypeGPIOInput(GPIO_PORTQ_BASE, GPIO_PIN_4);
    }
    else
```

```
    {
        //
        // Keep the default config for most pins used by USB.
        // Add a pull down to PD6 to turn off the TPS2052 switch
        //
        ROM_GPIOPinTypeGPIOInput(GPIO_PORTD_BASE, GPIO_PIN_6);
        MAP_GPIOPadConfigSet(GPIO_PORTD_BASE, GPIO_PIN_6, GPIO_STRENGTH_2MA,
                             GPIO_PIN_TYPE_STD_WPD);

    }

    //
    // PF0/PF4 are used for Ethernet LEDs.
    //
    if (bEthernet)
    {
        //
        // this app wants to configure for ethernet LED function.
        //
        ROM_GPIOPinConfigure(GPIO_PF0_EN0LED0);
        ROM_GPIOPinConfigure(GPIO_PF4_EN0LED1);

        GPIOPinTypeEthernetLED(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4);

    }
    else
    {

        //
        // This app does not want Ethernet LED function so configure as
        // standard outputs for LED driving.
        //
        ROM_GPIOPinTypeGPIOOutput(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4);

        //
        // Default the LEDs to OFF.
        //
        ROM_GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4, 0);
        MAP_GPIOPadConfigSet(GPIO_PORTF_BASE, GPIO_PIN_0 | GPIO_PIN_4,
                             GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);


    }

    //
    // PJ0 and J1 are used for user buttons
    //
    ROM_GPIOPinTypeGPIOInput(GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    ROM_GPIOPinWrite(GPIO_PORTJ_BASE, GPIO_PIN_0 | GPIO_PIN_1, 0);

    //
    // PN0 and PN1 are used for USER LEDs.
    //
    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    MAP_GPIOPadConfigSet(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1,
                         GPIO_STRENGTH_12MA, GPIO_PIN_TYPE_STD);

    //
    // Default the LEDs to OFF.
    //
    ROM_GPIOPinWrite(GPIO_PORTN_BASE, GPIO_PIN_0 | GPIO_PIN_1, 0);
}
```

```
//*****************************************************************************
//
//! This function writes a state to the LED bank.
//!
//! \param ui32LEDMask is a bit mask for which GPIO should be changed by this
//! call.
//! \param ui32LEDValue is the new value to be applied to the LEDs after the
//! ui32LEDMask is applied.
//!
//! The first parameter acts as a mask.  Only bits in the mask that are set
//! will correspond to LEDs that may change.  LEDs with a mask that is not set
//! will not change. This works the same as GPIOPinWrite. After applying the
//! mask the setting for each unmasked LED is written to the corresponding
//! LED port pin via GPIOPinWrite.
//!
//! \return None.
//
//*****************************************************************************
void
LEDWrite(uint32_t ui32LEDMask, uint32_t ui32LEDValue)
{

    //
    // Check the mask and set or clear the LED as directed.
    //
    if (ui32LEDMask & CLP_D1)
    {
        if (ui32LEDValue & CLP_D1)
        {
            GPIOPinWrite(CLP_D1_PORT, CLP_D1_PIN, CLP_D1_PIN);
        }
        else
        {
            GPIOPinWrite(CLP_D1_PORT, CLP_D1_PIN, 0);
        }
    }

    if (ui32LEDMask & CLP_D2)
    {
        if (ui32LEDValue & CLP_D2)
        {
            GPIOPinWrite(CLP_D2_PORT, CLP_D2_PIN, CLP_D2_PIN);
        }
        else
        {
            GPIOPinWrite(CLP_D2_PORT, CLP_D2_PIN, 0);
        }
    }

    if (ui32LEDMask & CLP_D3)
    {
        if (ui32LEDValue & CLP_D3)
        {
            GPIOPinWrite(CLP_D3_PORT, CLP_D3_PIN, CLP_D3_PIN);
        }
        else
        {
            GPIOPinWrite(CLP_D3_PORT, CLP_D3_PIN, 0);
        }
    }
```

```
    if (ui32LEDMask & CLP_D4)
    {
        if (ui32LEDValue & CLP_D4)
        {
            GPIOPinWrite(CLP_D4_PORT, CLP_D4_PIN, CLP_D4_PIN);
        }
        else
        {
            GPIOPinWrite(CLP_D4_PORT, CLP_D4_PIN, 0);
        }
    }
}

//*****************************************************************************
//
//! This function reads the state to the LED bank.
//!
//! \param pui32LEDValue is a pointer to where the LED value will be stored.
//!
//! This function reads the state of the CLP LEDs and stores that state
//! information into the variable pointed to by pui32LEDValue.
//!
//! \return None.
//
//*****************************************************************************
void LEDRead(uint32_t *pui32LEDValue)
{
    *pui32LEDValue = 0;

    //
    // Read the pin state and set the variable bit if needed.
    //
    if (GPIOPinRead(CLP_D4_PORT, CLP_D4_PIN))
    {
        *pui32LEDValue |= CLP_D4;
    }

    //
    // Read the pin state and set the variable bit if needed.
    //
    if (GPIOPinRead(CLP_D3_PORT, CLP_D3_PIN))
    {
        *pui32LEDValue |= CLP_D3;
    }

    //
    // Read the pin state and set the variable bit if needed.
    //
    if (GPIOPinRead(CLP_D2_PORT, CLP_D2_PIN))
    {
        *pui32LEDValue |= CLP_D2;
    }

    //
    // Read the pin state and set the variable bit if needed.
    //
    if (GPIOPinRead(CLP_D1_PORT, CLP_D1_PIN))
    {
        *pui32LEDValue |= CLP_D1;
    }
}
```

```
//*****************************************************************************
//
// Close the Doxygen group.
//! @}
//
//*****************************************************************************

#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include <math.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "driverlib/gpio.h"
#include "drivers/pinout.h"
#include "drivers/humiditysensor.h"
#include "drivers/altitudesensor.h"
#include "inc/hw_gpio.h"
#include "driverlib/pin_map.h"
#include "driverlib/rom.h"
#include "driverlib/rom_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/ssi.h"
#include "driverlib/i2c.h"
#include "utils/uartstdio.h"
#include "FreeRTOSConfig.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "timers.h"
#include "limits.h"
#include "string.h"
#include <time.h>

#define UART_COMM

TimerHandle_t xTimer1,xTimer2;
TaskHandle_t xHBTaskHandle;
QueueHandle_t HQueue, AQueue;
BaseType_t xHBTask;

volatile float humidity;
void AltitudeTask(void *pvParameters);
void HumidityTask(void *pvParameters);
void LoggerTask(void *pvParameters);

#ifdef DEBUG
void
__error__(char *pcFilename, uint32_t ui32Line)
{
}
#endif


struct Message
{
    uint8_t log_id;
    uint8_t log_level;
    float data;
    char timestamp[25];
```

```
    char c;
}xMessage;

#define ALT  0x01
#define HUMID  0X02

uint32_t g_ui32SysClock;

void ConfigureUART(void)
{

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    ROM_GPIOPinConfigure(GPIO_PA0_U0RX);
    ROM_GPIOPinConfigure(GPIO_PA1_U0TX);
    ROM_GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    UARTStdioConfig(0, 115200, g_ui32SysClock);
}

void uart7_init(void)
{
    //
    // Enable the GPIO Peripheral used by the UART.
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOC);

    //
    // Enable UART2
    //
    ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_UART7);

    //
    // Configure GPIO Pins for UART mode.
    //
    ROM_GPIOPinConfigure(GPIO_PC4_U7RX);
    ROM_GPIOPinConfigure(GPIO_PC5_U7TX);
    ROM_GPIOPinTypeUART(GPIO_PORTC_BASE, GPIO_PIN_4 | GPIO_PIN_5);

    //
    // Initialize the UART for console I/O.
    //
    //UARTStdioConfig(0, 9600, g_ui32SysClock);
    ROM_UARTConfigSetExpClk(UART7_BASE, g_ui32SysClock, 9600,
                                (UART_CONFIG_WLEN_8 | UART_CONFIG_STOP_ONE |
                                 UART_CONFIG_PAR_NONE));
}

void uart7_send(const uint8_t *pui8Buffer, uint32_t ui32Count)
{
    // Loop while there are more characters to send.
    while(ui32Count--)
    {
        // Write the next character to the UART.
         UARTCharPut(UART7_BASE, *pui8Buffer++);
    }
}

void
InitSPI3(void)
```

```
{

    SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI3);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

    //
    // Configure the pin muxing for SSI3 functions on port H4, H5, H6 and H7.
    // This step is not necessary if your part does not support pin muxing.
    //
    GPIOPinConfigure(GPIO_PF3_SSI3CLK);
    GPIOPinConfigure(GPIO_PF2_SSI3FSS);
    GPIOPinConfigure(GPIO_PF0_SSI3XDAT1);
    GPIOPinConfigure(GPIO_PF1_SSI3XDAT0);

    GPIOPinTypeSSI(GPIO_PORTF_BASE, GPIO_PIN_3 | GPIO_PIN_1 | GPIO_PIN_0 |
                   GPIO_PIN_2);

    SSIConfigSetExpClk(SSI3_BASE, g_ui32SysClock, SSI_FRF_MOTO_MODE_2,
                       SSI_MODE_SLAVE, 100000, 8);

    SSIEnable(SSI3_BASE);
}

void spi3_send(const uint8_t *pui32Data, uint32_t ui32Index)
{
    while(ui32Index--)
    {
        SSIDataPut(SSI3_BASE , *pui32Data ++ );
    }
    while(SSIBusy(SSI3_BASE))
    {
    }

}

void vTimerCallback1( TimerHandle_t xTimer1 )
{

    struct Message px1Message;
    float altitude;
    time_t a = time(NULL);
    char* temp=ctime(&a);
    strcpy(px1Message.timestamp,temp);
    altitude = get_altitude();
    //Tick_Count = xTaskGetTickCount();
    px1Message.data = altitude;
    px1Message.log_id = 1;
    px1Message.log_level = 1;
    //strcpy( pxMessage.logstring, "Anay here");

    /*
    AQueue = xQueueCreate( 10, sizeof( struct Message ) );
    xQueueSend( AQueue, &pxMessage, 10 );
    */
    //xQueueSendToBack( HQueue, &px1Message, 10 );
    //UARTprintf("%d", (int)altitude);

    if(altitude < 0 || altitude > 8850)
    {
        xTaskNotify( xHBTaskHandle,ALT , eSetBits );
    }
```

```
    else
    {
        xQueueSendToBack( HQueue, &px1Message, 10 );
    }
}

void vTimerCallback2( TimerHandle_t xTimer2 )
{

    struct Message pxMessage;
    time_t a = time(NULL);
    char* temp=ctime(&a);
    strcpy(pxMessage.timestamp,temp);
    float humidity1 = get_humidity();
            if((humidity - humidity1) > 10.0 || (humidity1 - humidity) > 10.0)
                    {
                        humidity1 = humidity;
                    }
            pxMessage.data = humidity;
            humidity = humidity1;

    //xQueueSendToBack( HQueue, &pxMessage, 10 );
    pxMessage.log_id = 2;
    pxMessage.log_level = 1;
    //xQueueSendToBack( HQueue, &pxMessage, 10 );

    if(humidity < 0 || humidity > 100)
        {
            xTaskNotify( xHBTaskHandle,HUMID , eSetBits );
        }

    else
    {
        xQueueSendToBack( HQueue, &pxMessage, 10 );
    }
}
void AltitudeTask(void *pvParameters)
{

        // Turn on LED 1
        xTimer1 = xTimerCreate("timer1",pdMS_TO_TICKS( 500 ),pdTRUE,( void * )
0,vTimerCallback1);
        xTimerStart( xTimer1, 0 );
        for(;;) ;

}

void HumidityTask(void *pvParameters)
{

        // Turn on LED 1
        xTimer2 = xTimerCreate("timer2",pdMS_TO_TICKS( 500 ),pdTRUE,( void * )
0,vTimerCallback2);
        xTimerStart( xTimer2, 0 );
        for(;;) ;

}

void LoggerTask(void *pvParameters)
{
    struct Message pxRxedMessage;
    while(1)
```

```
    {

        if( xQueueReceive( HQueue, &( pxRxedMessage ), portMAX_DELAY ))
                {

                        if(pxRxedMessage.log_id == 1)
                               {
                                 UARTprintf("Log ID = %d  ",pxRxedMessage.log_id);
                                 UARTprintf("Altitude is %d\n",
(int)pxRxedMessage.data);
                                 UARTprintf("Timestamp: %s
\n",pxRxedMessage.timestamp );
                                 UARTprintf("Log Level = %d
",pxRxedMessage.log_level);
                               }
                              else if(pxRxedMessage.log_id == 2)
                               {
                                  UARTprintf("Log ID = %d  ",pxRxedMessage.log_id);
                                 UARTprintf("Humidity is %d\n",
(int)pxRxedMessage.data);
                                 UARTprintf("Timestamp: %s
\n",pxRxedMessage.timestamp );
                                 UARTprintf("Log Level = %d
",pxRxedMessage.log_level);
                                }

                        #ifdef UART_COMM
                        uart7_send((uint8_t *)&pxRxedMessage,sizeof(pxRxedMessage));
                        #else
                         spi3_send((uint8_t *)&pxRxedMessage, sizeof(pxRxedMessage));
                       #endif
                }
        //SysCtlDelay(g_ui32SysClock / 2 / 3);
    }
}

void HBTask(void *pvParameters)
{
    uint32_t val_recv;
    struct Message hbmsg;
    time_t a = time(NULL);
    while(1)
    {
        xHBTask = xTaskNotifyWait(0, 0xFF, &val_recv, portMAX_DELAY);
        if(xHBTask == pdTRUE)
        {
            if(val_recv & 0x01)
            {
                UARTprintf("Error in Heartbeat from Altitude \n");

            char* temp=ctime(&a);
            strcpy(hbmsg.timestamp,temp);

            hbmsg.data = -2;

            hbmsg.log_id = 1;
            hbmsg.log_level = 2;
            xQueueSendToBack( HQueue, &hbmsg, 10 );
            }

            if(val_recv & 0x02)
            {
```

```
            UARTprintf("Error in Heartbeat from Humidity \n");

            char* temp=ctime(&a);
            strcpy(hbmsg.timestamp,temp);

            hbmsg.data = -2;

            hbmsg.log_id = 2;
            hbmsg.log_level = 2;
            xQueueSendToBack( HQueue, &hbmsg, 10 );
            }
        }
    }
}

int main(void)
 {

    //
    // Run from the PLL at 120 MHz.
    //
    g_ui32SysClock = MAP_SysCtlClockFreqSet((SYSCTL_XTAL_25MHZ |
                SYSCTL_OSC_MAIN | SYSCTL_USE_PLL |
                SYSCTL_CFG_VCO_480), 120000000);

    //
    // Configure the device pins.
    //
    PinoutSet(false, false);

    ROM_GPIOPinTypeGPIOOutput(GPIO_PORTN_BASE, GPIO_PIN_1);

    ConfigureUART();

    uart7_init();


    InitSPI3();
    UARTprintf("UART Configured");

    ConfigureAltitude();

    ConfigureHumidity();
    HQueue = xQueueCreate( 10, sizeof( struct Message ) );

    SysCtlDelay(g_ui32SysClock / 2 / 3);
    humidity = get_humidity();

    /* freertos based code */

    xTaskCreate(AltitudeTask, (const portCHAR
*)"ALTITUDE_TASK",configMINIMAL_STACK_SIZE, NULL, 1, NULL);

    xTaskCreate(HumidityTask, (const portCHAR
*)"HUMIDITY_TASK",configMINIMAL_STACK_SIZE, NULL, 1, NULL);

    xTaskCreate(LoggerTask, (const portCHAR *)"LOGGER_TASK",configMINIMAL_STACK_SIZE,
NULL, 1, NULL);

    xTaskCreate(HBTask, (const portCHAR *)"HB_TASK",configMINIMAL_STACK_SIZE, NULL,
1, &xHBTaskHandle);
```

```
    vTaskStartScheduler();

    UARTprintf("Ending Main");

    return 0;

}
```