SOFTWARE ARCHITECTURE

Multi-Controller, Data Acquisition and Remotely Logged Home Automation System

Project 2, APES

Sai Raghavendra Sankrantipati

Vishal Vishnani

DESCRIPTION

A multithreaded home automation system in which, BeagleBone(BBG) is used as a remote logger system while TIVA acts as a sensor hub. The sensor hub collects data values from different sensors and sends them to BBG over socket communication.

Overview of tasks

On TIVA

- Light sensor (APDS 9601) task retrieves lux values from the sensor using I2C communication
- Temperature sensor (DHT 22) task retrieves temperature values from the sensor using I2C communication.
- Humidity sensor (DHT 22) task retrieves temperature values from the sensor using I2C communication.
- Socket communication task which handles sending data from TIVA to Beaglebone
- Main task which creates and monitors all above tasks.
- A task that handles creating message API and payload.

On BEAGLEBONE

- Socket communication task which handles receiving data from TIVA.
- Logger task to log data to different files depending on what type of sensor (temperature, light, humidity).
- Decision task used to notify user about current state of system (light, dark, above or below humidity and temperature threshold values) and to set/reset the LEDS accordingly.
- Main task which creates and monitors all above tasks.
- A task that handles creating message API and payload.

Components

- Beaglebone Green - 1
- TIVA Development board – 1
- DHT 22 (Temperature and Humidty sensor) - 1
- APDS 9601 (Light sensor) – 1
- Linux OS on BBG
- Free RTOS on TIVA
- TIVA ware HAL library
- BSD Sockets API

Data structures

- On Beaglebone, we will use POSIX message queues for IPC and Pthread API (mutexes, semaphores and condition variables) for mutual exclusion, synchronisation and signalling.
- On Tiva, we will use FreeRTOS Queue Set API Functions for messaging.
- RTOS task notification API functions for inter-task notifications and FreeRTOS Mutexes for synchronisation.

Mechanisms

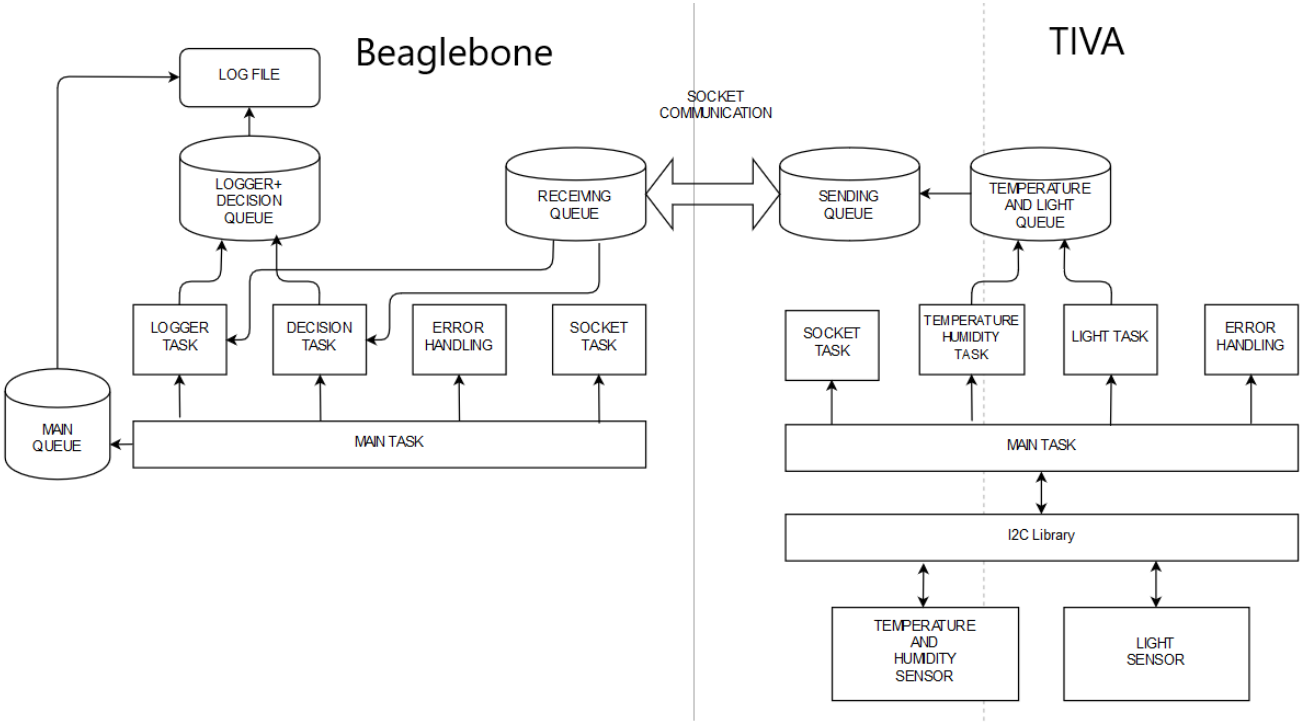
Userspace

- BSD Sockets API
- TIVA ware HAL library for interfacing with all sensors
- Pthread API (mutexes, semaphores and condition variables) for mutual exclusion, synchronisation and signalling

Kernel:

- USB Linux Kernel Driver (Future scope)

DIAGRAM


Beaglebone
TIVA
LOG FILE
SOCKET COMMUNICATION
LOGGER+ DECISION QUEUE
RECEIVING QUEUE
SENDING QUEUE
TEMPERATURE AND LIGHT QUEUE
LOGGER TASK
DECISION TASK
ERROR HANDLING
SOCKET TASK
MAIN QUEUE
MAIN TASK
SOCKET TASK
TEMPERATURE HUMIDITY TASK
LIGHT TASK
ERROR HANDLING
MAIN TASK
I2C Library
TEMPERATURE AND HUMIDITY SENSOR
LIGHT SENSOR

FUNCTION STUBS

1.RTOS

```
/* @brief Enum for level of log. Names are self explanatory*/
typedef enum{
	STARTUP,
	INFO,
	ALERT,
	CRITICAL
}log_level_t;

/* @brief Enum for types of source. Names are self explanatory*/
typedef enum{
	DHT22_TASK,
	APDS9601_TASK,
}source_t;

/* @brief Enum for types of log. Names are self explanatory*/
typedef enum{
	INIT,
	FAILURE,
	DATA,
	REQUEST,
	RESPONSE,
	HEART_BEAT
}log_type_t;

/* @brief Basic message struct for inter process communcaition.
	It can be used for communcaition within RTOS tasks and also for
	communication over sockets
	task_ticks gives the the count of ticks since VTaskScheduler was called
*/
typedef struct {
	long int task_ticks;
	log_level_t log_level;
	source_t src_id;
	log_type_t log_type;
	void * data;
}message_t;
```

```
QueueHandle_t queue = 0;

/* @brief This task communicates with Beaglebone Green using sockets
        This takes messages from dht22_task and apds9601_task
*/
void relay_task(void *p)
{
        message_t recv_mesg;
        //Recieve a message on Queue 'queue' into recv_mesg
        xQueueRecieve( queue,( void * )&recv_mesg, 0);

        /* @brief This function transmits messages over sockets to Beaglebone
                        It expects a acknowledgement from BBG. If it doesn't recieve ack.
                        function returns -1
          @param This function takes message struct as input
          @return return -1 on failure
        */
        transmit_message( &recv_mesg );
}

void dht22_task(void *p)
{
        message_t send_mesg;
        /* @brief This function initialises DHT22 sensor over I2C bus
          @return return -1 on failure
        */
        dht22_init();


        /* @brief This function writes command to control
                        register of DHT22 sensor over I2C bus
          @param takes uint16_t, a command to be written as input
          @return return -1 on failure
        */
        write_dht22Control( command );

        while(1) {

                /* @brief This function gets raw temperature from DHT22 and returns on converting.
                        @param takes a pointer to float where temperature will be returned.
                        @return return -1 on failure
                */
```

```
                get_temperature( *temp );

                /* @brief This function gets raw humidity from DHT22 and returns on converting.
                        @param takes a pointer to float where humidity will be returned.
                        @return return -1 on failure
                */
                get_humidity( *humidity );
                //send a message on Queue 'queue' from send_mesg
                xQueueSend( queue, (void *)&send_mesg, 0);
        }
}

void apds9601_task(void *p)
{
        message_t send_mesg;
        /* @brief This function initialises APDS9601 sensor over I2C bus
          @return return -1 on failure
        */
        apds9601_init();

        /* @brief This function writes command to control
                        register of APDS9601 sensor over I2C bus
          @param takes uint16_t, a command to be written as input
          @return return -1 on failure
        */
        write_APDSControl( command );
        while(1) {
                /* @brief This function gets raw luminosity from APDS9601 and returns on converting.
                        @param takes a pointer to float where luminosity will be returned.
                        @return return -1 on failure
                */
                get_luminosity( *lumen );
                //send a message on Queue 'queue' from send_mesg
                xQueueSend( queue, (void *)&send_mesg, 0);
        }
}

int main()
{
        /* Create a queue containing 10 messages of message_t type */
        qh = xQueueCreate(10, sizeof(message_t));
        //Create tasks
        xTaskCreate(relay_task, "task1", STACK_BYTES(BYTES), 0, 1, 0);
```

```
        xTaskCreate(dht22_task, "task2", STACK_BYTES(BYTES), 0, 1, 0);
        xTaskCreate(apds9601_task, "task3", STACK_BYTES(BYTES), 0, 1, 0);
        vTaskStartScheduler();
}
```

2. Beaglebone Green

```
/* @brief Enum for level of log. Names are self explanatory*/
typedef enum{
	STARTUP,
	INFO,
	ALERT,
	CRITICAL
}log_level_t;

/* @brief Enum for types of source. Names are self explanatory*/
typedef enum{
	DHT22_TASK,
	APDS9601_TASK,
}source_t;

/* @brief Enum for types of log. Names are self explanatory*/
typedef enum{
	INIT,
	FAILURE,
	DATA,
	REQUEST,
	RESPONSE,
	HEART_BEAT
}log_type_t;

/* @brief Basic message struct for inter process communcaition.
	It can be used for communcaition within RTOS tasks and also for
	communication over sockets
	task_ticks gives the the count of ticks since VTaskScheduler was called
*/
typedef struct {
	long int task_ticks;
	log_level_t log_level;
	source_t src_id;
	log_type_t log_type;
	void * data;
}message_t;

/* @brief This task reads a message of struct message_t from and logs it into a file
		The report includes status of the server, new connection requests
*/
void *log_task()
```

```
{
        message_t rmsg;

        while(1){
                //recieve message from the queue
                mq_receive(queue, (char*)&rmsg, sizeof(rmsg1), NULL);

                /* @brief In main, all tasks and queues are created
                        @param A pointer to message structure and a file descriptor of size int
                        @return returns -1 on failure
                */
                log_file(&rmsg, fd);
                }
}

/* @brief This task reads a message of struct message_t from queue and takes decision accordingly
        This task can issue and recieve remote API calls to the TIVA board
*/
void *decison_task()
{

}
/* @brief In main, all tasks and queues are created and makes sure all tasks are alive and running
                 In case of error it logs to the file and activate User LEDS
  @param In command line, it takes file name as an argument
  @return returns -1 on failure
*/
int main(int argc, char *argv[]){
}
```