REPORT

AESD: PROJECT -I

Project Partners

Raj Lavingia

Yash Gupte

INDEX

Topic *Page No.*

Project Overview

Design and Implement **"smart" environment** monitoring device using two sensors called temperature sensor and light sensor.
Temperature sensor (TMP102) and **light sensor(APDS-9301)** will be used for constantly monitoring the temperature of the device and for detection / sensing of the light in the area respectively.
The project will be implemented on **BeagleBone Green Board**.
The sensors will be interfaced with I2C communication method and continuously log data into a log file of both the sensors.

For this project, we have decided to compile 1 **Parent Thread** which is the **Main Task**.
The main task is spawned into 5 different child threads according to the requirement mentioned in the project.

> Note : There is only 1 bus where read and write of data takes place and since there is only 1 bus common for both the sensors a need arises for the forking of threads.
>
> Note : **It is very important to exit the program gracefully**. In order to do that, the Main thread will continuously monitor all the following threads on a certain time period. If the threads are not able to respond(even 1 of them) in a given time interval then the Main thread will exit gracefully closing all the child threads.
> For making such tasks, we have planned to use **pthreads.**

Thread Description

1. **Temperature Sensor Task** - This thread will be used for continuously for monitoring the Temperature in either Celsius or Fahrenheit through an I2C bus communication connected between a Microcontroller and the Temperature Sensor (TM102)
2. **Light Sensor Task** - This thread will be used for continuously for monitoring the Light and getting the lux values through an I2C bus communication connected between a Microcontroller and the Light Sensor (APDS-9301)
3. **Socket Task** - In this thread, we will have a TCP server running continuously and which accepts connection from the client which can ask for various requests like sensor data.
4. **Logging task** - This thread is very much useful for logging the data with a timestamp into a file which has been acquired from both the sensors on a continuous basis.

Block Diagram

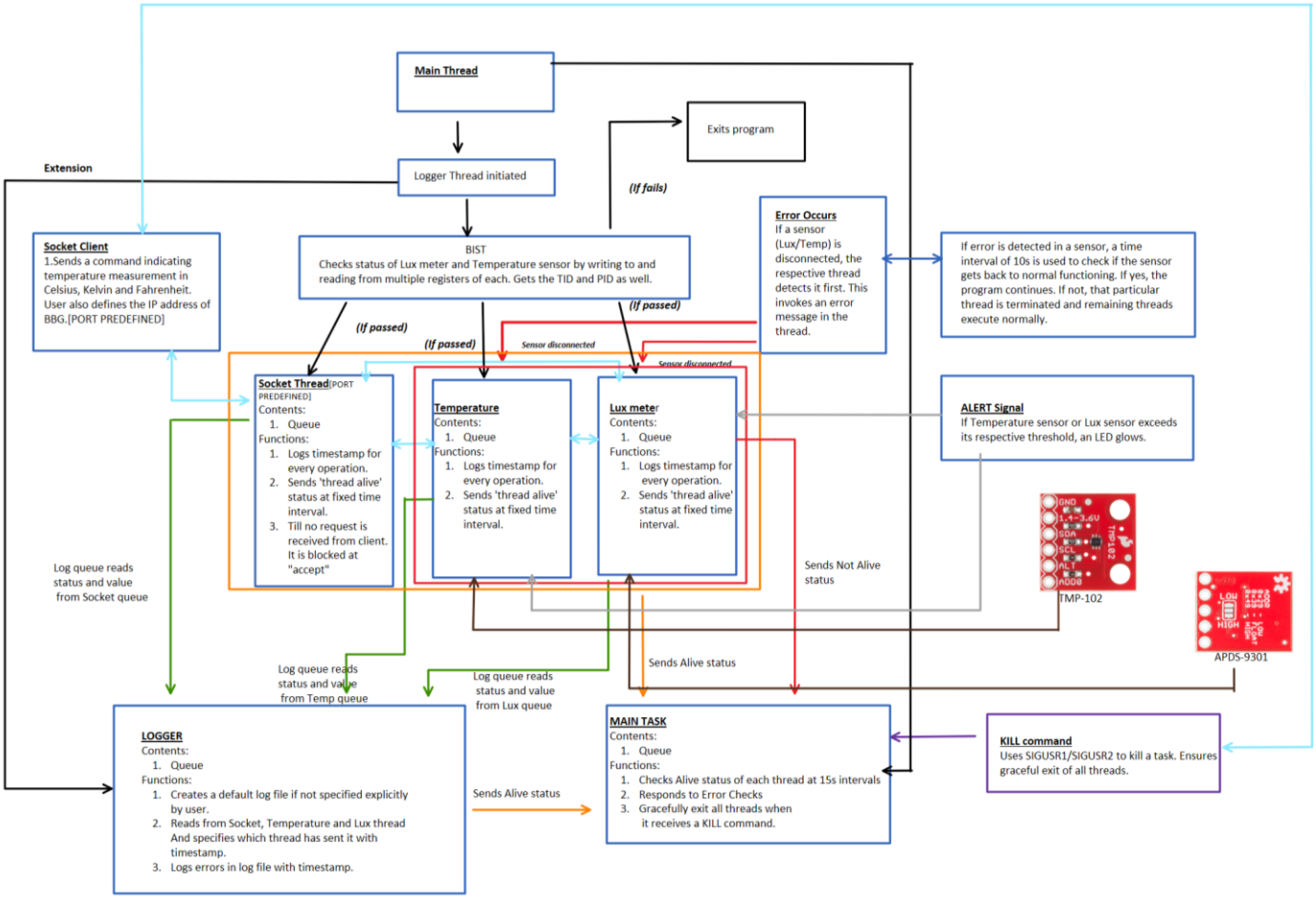

Fig. Block Diagram

BIST Description

Build In Self Test is used to check if Lux sensor and Temperature sensor are working. This is tested by using I2C communication read from register and write to register methods.

If BIST passes for both temperature and lux sensor, only then a separate thread for each lux sensor, temperature sensor and socket is created. If BIST fails for even one of the sensors, main thread exits gracefully without any retries.

BIST steps for Temperature sensor:

1. Temperature sensor is initialized.
2. Base registers are set.
3. All the remaining registers are written.
4. All the registers are read to confirm written values.
5. Base register is reset in order to measure temperature.
6. Once, all these tests are done, “BIST passed message from temperature sensor” is put on the logger queue which prints to text file.

BIST steps for Lux sensor:

1. Lux sensor is initialized.
2. Base registers are written.
3. All the remaining registers are written.
4. All the registers are read to confirm written values.
5. Base register is set to 0X03. To measure light intensity.
6. Once, all these tests are done, “BIST passed message from lux sensor” is put on the logger queue which prints to text file.

API Description

Temperature Thread:

1. uint8_t base_reg_write(uint8_t* buffer_value,int buffer_bytes);// Common base register for writing to Thigh,Tlow and config.
2. void * TempThread(void * args);// Individual temp queue is made and communicated with all the remaning threads. Also checks for kill command from USR1 or USR2.If socket client asks for temperature in any unit, this thread returns that specific unit (C,F,K)
3. uint8_t base_reg_read(uint8_t *buffer_value,int buffer_bytes);// Common base register for reading Thigh,Tlow and config.
4. uint8_t main_write_register(uint8_t register_addr, uint16_t desired_val); //Register addr must be between(0x01-0x03).Common base funciton for all registers (writing purpose), sends a buffer to the register through file descriptor.
5. uint8_t main_read_register(uint8_t register_addr, uint8_t* desired_val); //Register addr must be between(0x01-0x03).Common base funciton for all registers (writing purpose), sends a buffer to the register through file descriptor. Reads whole buffer through file-descriptor and prints two 8 bit numbers.(LSB,MSB)
6. uint8_t all_registers_check(void); // Thigh and Tlow are written and read according to threshold set.
7. uint8_t config_register_temperature(void); // Bits of config reg are written and read for purposes like shutdown mode, fault bits - em mode and cr(conversion rate) mode
8. uint8_t get_temp(float *t_data); // Calculations are done in order to get a temp value in float in C,K or F from adc. Also checks id measured temo is above or below threshold.
9. uint8_t temp_initial_sensor(void); // Starts temp sensor by openening I2C bus and checks slave addresses of I2C.
10. uint8_t BIST_Temp_Check(void); // BIST is performed.

Lux thread:

1. uint8_t lux_common_write(uint8_t* buffedesired_value,int buffer_bytes); //function for writing to base register.
2. uint8_t lux_common_read(uint8_t *buffedesired_value,int buffer_bytes); //function for reading from base register.
3. uint8_t write_pointer(uint8_t* x);
4. uint8_t lux_read_reg(uint8_t* x,uint8_t bytes); //Common custom function for reading a desired value from a specific address. Returns 0 if value is read successfully.
5. uint8_t lux_write_reg1(uint8_t* x,uint8_t bytes);
6. uint8_t lux_write_reg(uint8_t* x); //common custom function for all registers writing a desired value to a specific address. Allows register addresses from (0x07 – 0x09) and 0x0b.
7. uint8_t Word_Data_Register (uint8_t x);
8. uint8_t Command_Write_Register(uint8_t x);
9. uint8_t lux_write_register(uint8_t register_addr, uint8_t desired_val);
10. uint8_t lux_read_register(uint8_t register_addr, uint8_t* desired_val);
11. uint8_t custom_test_lux_config(void); //Base register address is set to 0x03(control reg start address) timing register gain and integration time are set, interrupt control register is written and read, threshold low-high, thresh high-low, threshold low-low and threshold high-high are written and read.
12. uint8_t get_lux(float *lux_final_value); //function used to get lux value from data[0] and data[1] registers by finding the ratio of data[0] and data [1]. The calculation for finding lux are taken from datasheet.
13. uint8_t day_night(float *tem); //if lux < 100 Night else if > 100 Day
14. uint8_t lux_initial_sensor(void); // Init by I2C bus opening, ioctl is sued for multiple bus creation.
15. uint8_t LuxThread_Init(void); //BIST for lux in which sensor is initialized, all registers are initialized successfully, lux, temp and socket threads are not created.

Socket Overview:

Sequence of steps to initiated Server-Client communication.
Predefined address of client: 10.0.0.227 predefined port: 8000.
Predefined address of server: 127.0.0.1. predefined port: 8000.

1. Server will be started on BBG.
2. Server will be blocking at 'accept' command, waiting for client to bond. Until it is not bonded it will wait for accept from the client. When an external signal is given from the client which running on host machine by giving a parameter of "Celsius=25, Fahrenheit=75 and Kelvin=300" to get temperature and "Lux value" of light sensor, server and client bond and server acts according to request of the client. Also, client sends a full structure containing string and integer in order to compare the status and meanwhile, client also reads the structure which is sent, from the server.
3. Client code will be exited gracefully, while the server is still running.
4. Timestamp is printed on both ends.
5. Whenever, external request comes from the client, the request message is printed in the log file.

Log API:

2. void log_initialization(char* path_defined_user) // In this function, a default filename will be opened with a write command so that every time whenever the system is run over and over new file is created without appending and TID of this particular log thread is obtained in this function.
3. void filelogging(char* path_defined_user, MsgStruct* Message) //Determines the data's thread. And logs the data into the log file according to the message received from the queue.

Main API:

In main function if BIST passes, main thread is spawned into temp, lux and socket. All threads are created and joined. Even if a single thread is created, the program will continue to run.

If an external signal from USR1/USR2 comes, then main thread will kill all 4 threads and it will exit gracefully.

If some thread stops working due to technical fault then it will wait for 10s, before killing that thread. After 10s only that thread will be terminated.

Unit Testing Functions

Unit tests have been performed on:

1. Temperature sensor
2. Lux sensor
3. Logging thread

The functions used are the same as the ones defined in the APIs.

References

1. Queue: https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/
2. Socket: https://www.geeksforgeeks.org/socket-programming-cc/
3. Mutex: https://www.geeksforgeeks.org/mutex-lock-for-linux-thread-synchronization/
4. Errno: http://man7.org/linux/man-pages/man3/errno.3.html
5. TMP-102 datasheet: http://www.ti.com/lit/ds/symlink/tmp102.pdf
6. APDS-9301 datasheet: http://www.farnell.com/datasheets/1816958.pdf