Advanced Practical Embedded Software Design -- Project 1 Report

Team Members:

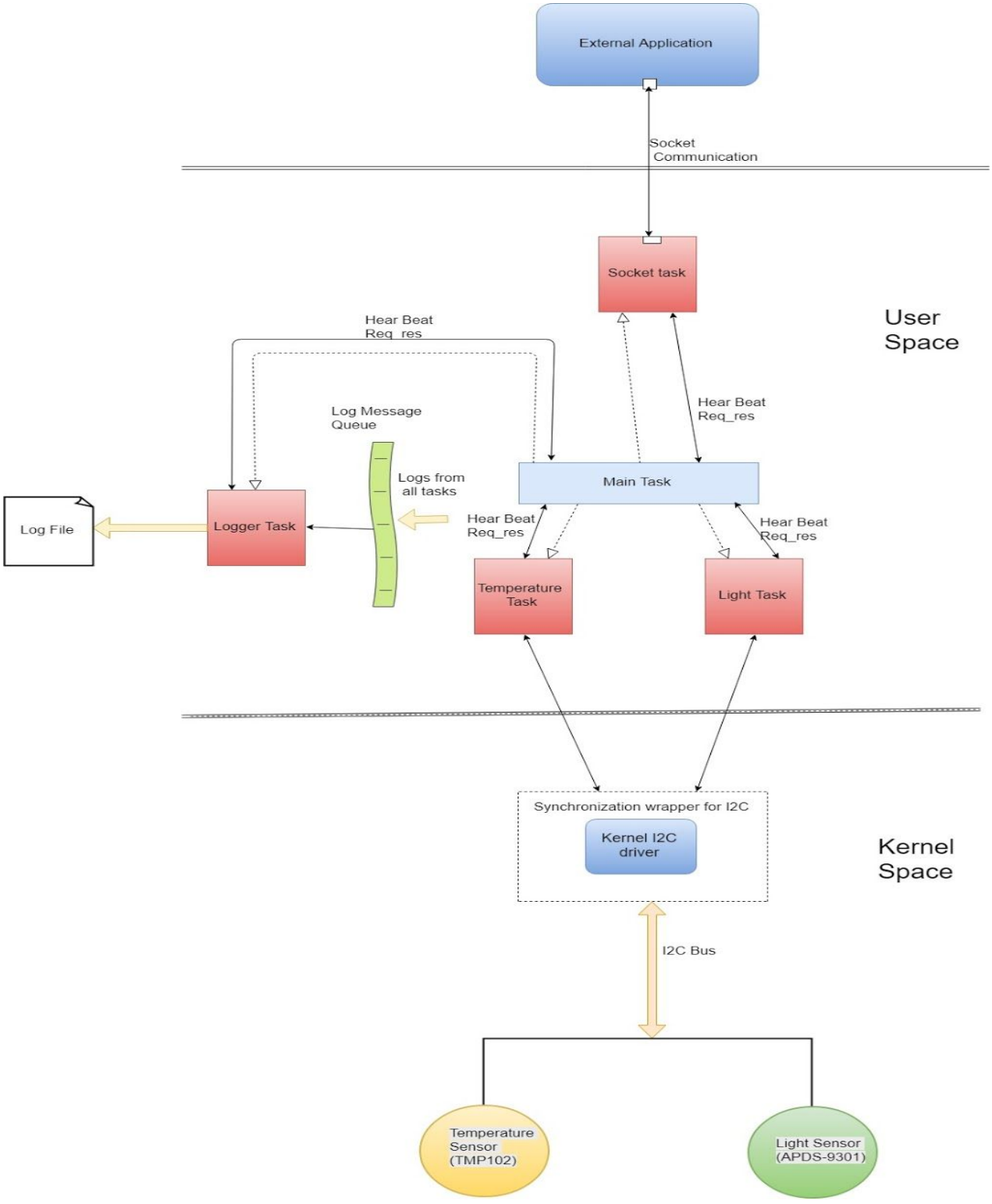
Pavan Dhareshwar

Sridhar Pavithrapu

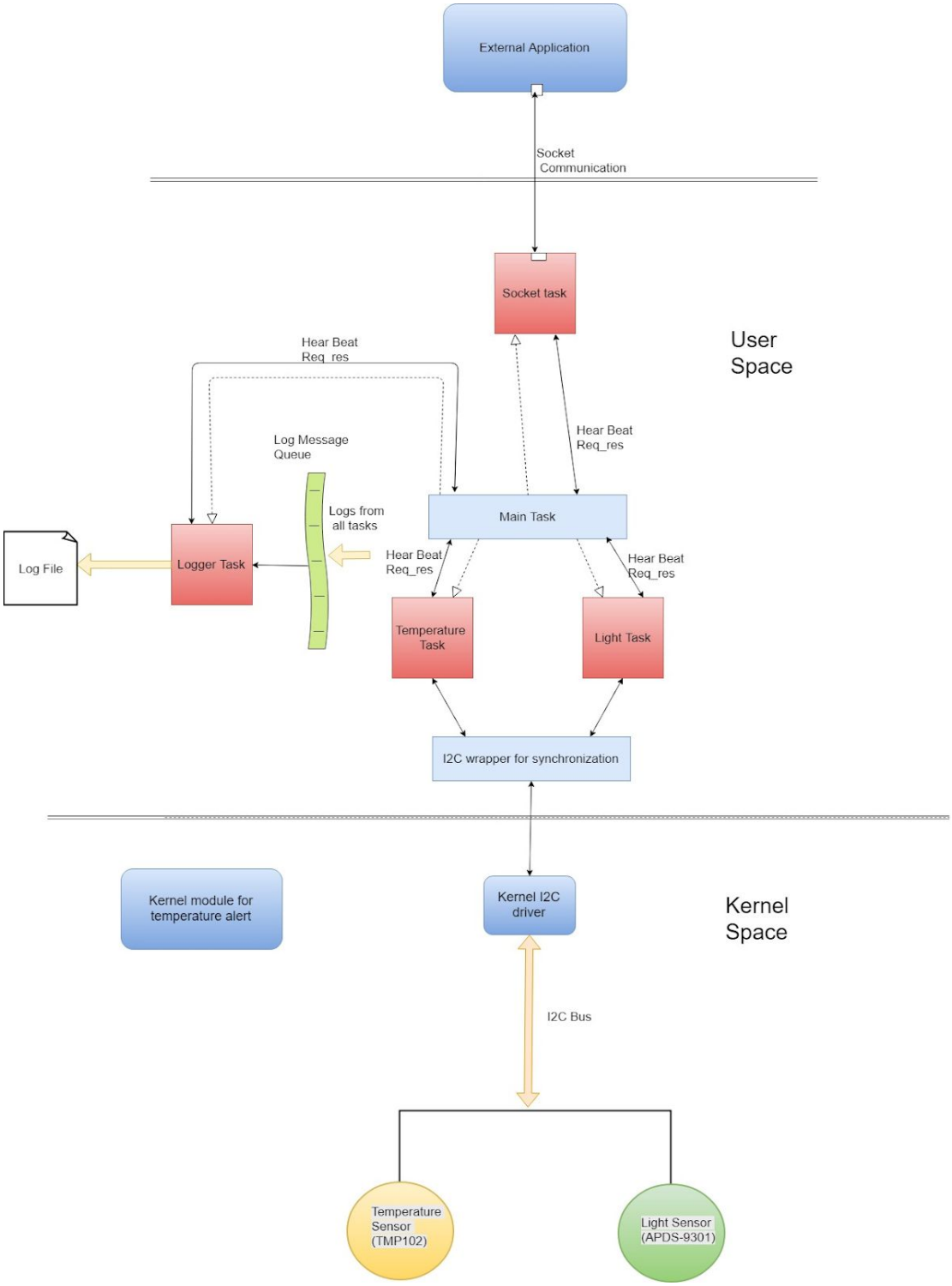
17th March 2018

Git Repository: https://github.com/pavandhareshwar/APES_Project

INITIAL SOFTWARE ARCHITECTURE



UPDATED SOFTWARE ARCHITECTURE



ARCHITECTURE DESCRIPTION

The system will have a main task that will be responsible for spawning the following four tasks.

1. Temperature task: This is a user-level task primarily responsible for interacting with the wrapper interface to configure the temperature sensor and read the current temperature. The other functionalities of this task include:
    a. This task also performs heart-beat message to main task, to make sure that the task is running continuously without any error.
    b. Logs the data whenever temperature data is read from the sensor.
    c. It also supports getting requests from external application and sends back the response accordingly.
2. Light task: This is another user-level primarily responsible for interacting with the kernel I2C wrapper to configure the light sensor and read the current lux value.The other functionalities of this task include:
    a. This task also performs heart-beat message to main task, to make sure that the task is running continuously without any error.
    b. Logs the data whenever lux data is read from the sensor.
    c. It also supports getting requests from external application and sends back the response accordingly.
3. Logger task: This is also a user-level task that is responsible to log all the messages received from other tasks like main, light, temperature and socket tasks information into a file.
4. Socket task: This is also a user-level task that will be responsible for getting requests from an external application (these requests include requests for both temperature and light tasks). Once the request is received, it is redirected according to either temperature or light task and the response is sent back to the external application.

Once the main task has spawned these subtasks, it will sleep and periodically wake up to request the status of each sub-tasks (the heart-beat) to ensure that all are up and running. If one or more tasks don't respond with an 'alive' response (no heart-beat), the main task will interpret this task(s) as not-running anymore and logs into the logger task for further debugging. The request-response messaging scheme between the main thread and the sub-tasks will be named and referred to by 'HB_req_rsp' until a definitive IPC scheme is finalized. Here message queue is to receive the messages from other tasks and logging into the file is performed by the logger task. Each task will create their own threads for heart-beat operation, receiving requests from external application and another one to log the sensor data/message to logger task.

In addition, for the extra credit a kernel module is developed. The functionality of the kernel module is to generate a interrupt request on rising edge of alert pin on temperature sensor. Once the interrupt is generated LED will glow on one of the GPIO ports registered in the kernel module.

API DESCRIPTION AND FUNCTIONALITY

Temperature Task:

As mentioned in the short introduction, the temperature task will be responsible for interacting with the wrapper to access I2C driver and perform read and write operations on the temperature sensor to configure the registers and get the temperature data.

The temperature task will provide the following set of API's:

int temp_sensor_init()

This API will perform all the initializations necessary to access the temperature sensor. The initialization includes setting up the configuration registers for powering on the sensor.

float read_temperature_data_register()

This API will be used to read the current temperature value from the temperature sensor. Additionally, a flag could be provided that would specify the format in which the temperature data needs to be given back (format is either celsius or fahrenheit).

int write_config_register_em(int res)

This API will be used to set the resolution of the temperature sensor to the value specified by 'res' argument.

Light task:

The light task will be responsible for interacting with the kernel synchronization interface for I2C driver and perform read and write operations on the light sensor to configure the registers and get the lux data.

The light sensor will provide the following set of API's:

int light_sensor_init()

This API will perform all the initializations necessary to access the light sensor.

int get_lux_data()

This API will be used to read the current lux value from the light sensor.

Logger task:

The logger task is responsible for logging all the required information to a log file. Since only the logger task has access to the log file, we will have a message queue from which the logger task will read the messages and logs them to a file. The remaining tasks will write to this message queue, if they want the respective information to be logged to the log file.

Socket task:

The socket task is responsible for handling request from an external application that could be a request for temperature task or light task. Since the socket task has to listen for requests from an external application, it will create a socket and wait for requests on that socket. Once it gets a request, it decodes the request and forwards it to either temperature or light task. Once the request is processed by the concerned task, the response is obtained and sent to the external application.

I2C wrapper:

int i2c_read()

This API is used to provide synchronized access to I2C bus to read the data from either temperature sensor or light sensor. It will also print the information to kernel logs.

int i2c_write()

This API is used to provide synchronized access of I2C bus to configure and to write command registers of either temperature sensor or light sensor. It will also print the information to kernel logs.

USER SPACE v/s KERNEL SPACE: What goes where?

The main task and the sub-tasks which the main task creates all operate in the user-space, communicating with each other via some form of IPC mechanism like a message queue. The wrapper for the I2C driver that ensures that the access to I2C bus in synchronization. The temperature task and light task will communicate with the kernel I2C driver with synchronization wrapper to either read/write data from temperature sensor or light sensor.

Kernel module is developed for the temperature extra credit.

References:

http://derekmolloy.ie/beaglebone/

http://wiki.seeed.cc/BeagleBone_Green/

http://tldp.org/LDP/lpg/node7.html

http://derekmolloy.ie/kernel-gpio-programming-buttons-and-leds/