ADVANCED EMBEDDED SYSTEM DEVELOPMENT

PROJECT – 1

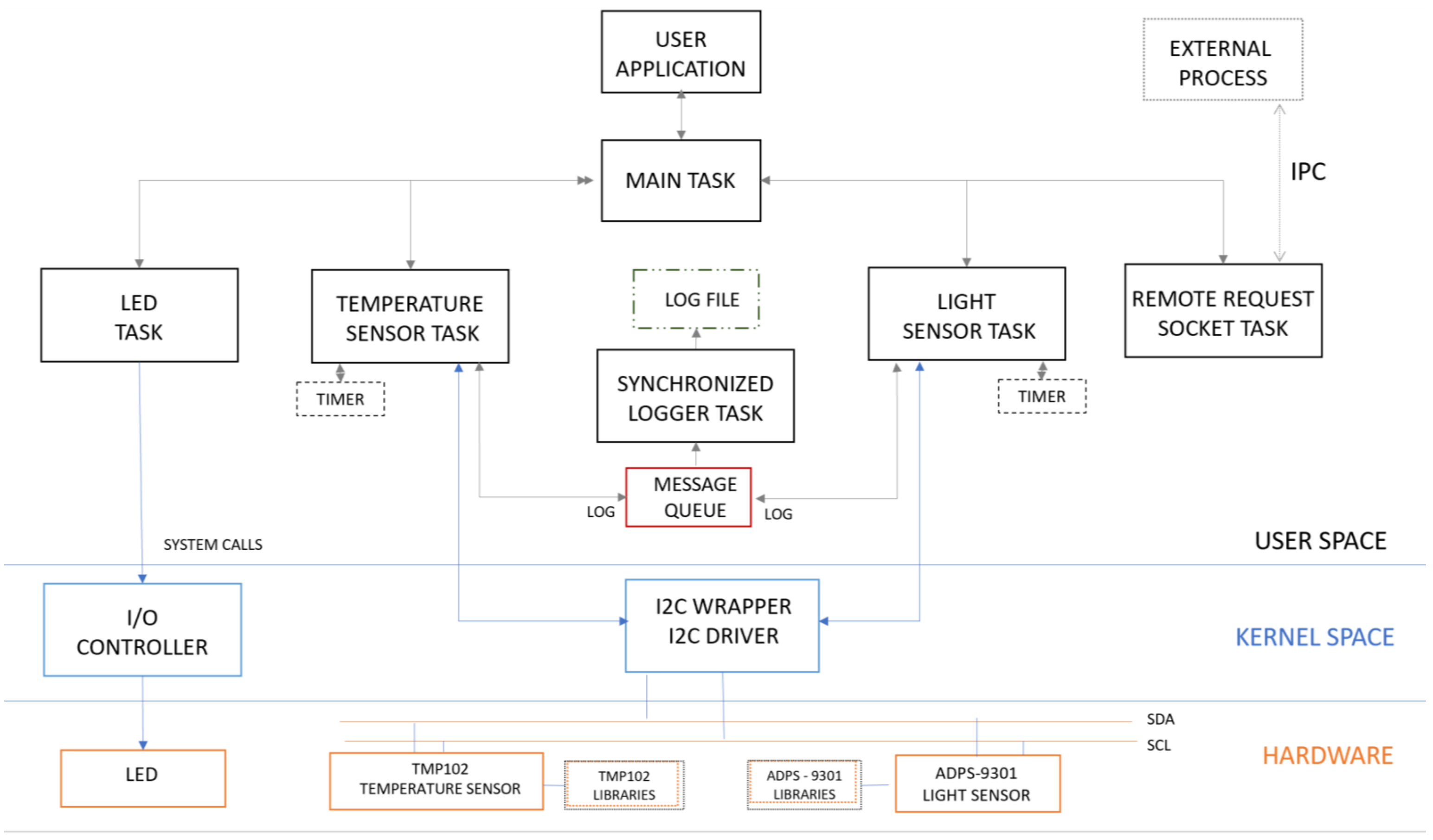
SYSTEM & ARCHITECTURE DOCUMENT

Deepesh Mahendra Sonigra

Madhumitha Tolakanahalli Pradeep

Github Repository : https://github.com/madhumithatp/Environment-Monitoring-Device

ARCHITECTURE DIAGRAM



APPLICATION TASKS

> MAIN TASK

The Main Task is the parent thread, implemented using asynchronous threading strategy to manage child threads. The Main Task performs following functions -

- Child Thread Management Function
- LED Task

The main task creates threads that perform the following functions –

- Temperature Sensor Task
- Light Sensor Task
- Synchronized Logger Task
- Remote Request Socket Task

CHILD THREAD MANAGEMENT FUNCTION

> The parent thread keeps the track of children tasks to check if they are alive and running on specified time intervals or dead and stuck for a specific timeout.
> This can be achieved by using Heartbeat notification from all child threads to the parent thread using request response mechanism.
> When requested to stop the task, the main task should clean up everything, issue exit commands to the child threads and terminate gracefully.

LED TASK

> This task is concerned notifying the user of error conditions encountered during the execution of the program (Example, missing sensor, abnormal child thread behaviour etc)
> The error is reported on the console and indicated using USR LED

TEMPERATURE SENSOR TASK

> This task is concerned with the Temperature Sensor TMP102 with I2C. It wakes up at a regular intervals to collect data from the sensor.
> It consists of read and write functions for sensor registers and modify configuration registers.
> Temperature data is available in binary format. This task performs conversion to decimal format. The temperature is to be made available in Celsius, Kelvin and Fahrenheit.
> Sample API list

```
TMP102_Init();              //Initial Set and Configuration
TMP102_BIST();              // Built in Start-up test
TMP102_ReadRegister(int * regptr);      //Read a register value
TMP102_WriteRegister(int * regptr);     //Write a register value
TMP102_BinToDecConv(uint * bin, int * dec); //Convert Binary to Decimal Value
TMP102_OutputTemperature(int * temp);        //Read a temperature
```

LIGHT SENSOR TASK

> This task is concerned with the Light Sensor APDS-9301 Light Sensor. It wakes up at regular intervals to collect data from the sensor.
> Interfaces are created for read and write for registers in 8-bit and 16-bit mode. This includes Read/Write functions for registers, modify Control Registers and read LUX values. Based on the LUX value, the task decides if the state is LIGHT or DARK.
> The task includes equations to read the actual Luminosity from the LUX Data ADC. It reads two types of light data – Visible Light and IR Light using Channels 0 & 1 correspondingly.
> Sample API List

```
APDS9301_Init();                    //Initial Set and Configuration
APDS9301_BIST();                    // Built in Start-up test
APDS9301_ReadRegister(int * regptr);   //Read a register value
APDS9301_WriteRegister(int * regptr);  //Write a register value
APDS9301_GetState(uint * LUX); //Get state light/dark
APDS9301_OutputLuminence(int * temp);        //Read Luminance Value
```

I2C WRAPPER FUNCTION

> This function is used to provide exclusive access to I2C bus to either Temperature Sensor Task or Light Sensor Task.

> Sample API List

```
I2CWrapper_BusStatus();          //Return status of I2C bus if busy or not
I2CWrapper_ReadTemperature();                    //To read temperature
I2CWrapper_ReadLuminence();            //To read luminence
```

SYNCHRONIZED LOGGER TASK

> This task is concerned with logging data from various on-board sources. As this interface is going to be accessed by multiple sources, synchronization is needed to maintain data consistency. In this project, data protection is achieved using **Message Queue.**

> The log file path and file name are configurable. This is accomplished using command line options.

> Once the process is requested/forced to close, it should close all file handles and terminate gracefully.

> Each log entry contains the following information: timestamp, log level, logger source ID, Log Message

REMOTE REQUEST SOCKET TASK

> This task acts as an interface to external programs and enables them to request sensor data.

> The Remote Request Socket Task accepts socket request from another process/task and makes API calls to sensor tasks for data.

> A second process is created to create a socket connection with this task to make remote requests.

> MESSAGE APIs

Inter-thread messaging is implemented using individual Message Queues. Messages being implemented are –

> Heartbeat Notification from all threads to main task: Implemented as a ping from the main task at regular intervals to ensure child threads are alive. If heartbeat messages from Main receive no response, the child threads are terminated after a time-out.

> Startup Tests: Each task should run an initialization routine and report the status to the main task.

> Data Request to Sensor Tasks: Tasks should communicate with Temperature and Light Sensors to be able to perform various functions like Read Temperature/ Light, modify control registers etc.

> Log Messages

> Error Reporting

> Child Thread Closing: Main task should be able to send the child tasks a request to close

BUILT-IN SELF TESTS(BIST)

Once a task is started, a few tests need to be performed to verify if the hardware and software are working. Tests include

- Temperature Sensor BIST to ensure correct working of I2C and hardware
- Light Sensor BIST to ensure correct working of I2C and hardware
- Child Thread BIST to ensure child threads are up and running correctly

After successful completion of each BIST, log message needs to be sent to the logger. If any test fails, error code is logged and USR LED is turned on.

UNIT TESTS

Software must be tested using CUnit Unit Test Framework for functions like –

- Inter thread communication
- Logger
- Temperature conversions
- Light Sensor conversions