




Basic APIs Used for different tasks				
Temperature Task	Light Task	Socket Task	Logger Task	BIST Task
float getTemp(unit_enum unit)	float getLight_8BIT()	float RequestTemp()	Log_T loggerEnqueue(...)	Int testTempSensor()
	float getLight_16BIT()	float RequestLight()	Log_T loggerDequeue(...)	Int testLightSensor()
		float RequestBoth()		Int testThread_Stat()
				Int testLogger_Stat()

Software design documentation

-by Shreya Chakraborty

Understanding of Project 1:

- ➔ Project 1 requires BBG to be connected to 2 sensors, temperature sensor TMP102 and Light sensor APDS-9301 via a single I2C bus. The project has been divided into several tasks. The Main task spawns all the children task. It ensures the tasks are alive and run to completion as well as cleanly exit each task. It spawns 5 tasks namely -> Temperature Task, Light task, Socket Task, Built in Self-test Task and Logger Task.
- ➔ The relation between each task and connection to various modules are as given in the software architecture diagram. All boxes having black colored arrow () pointed to is are threads spawned by the main. The various modules in each task are colored coded and so are the arrows.
- ➔ All the tasks periodically send message to the main task (). Also they send message to the external client asking for data on demand basis ().
- ➔ The entire project is to be implemented in a layered architecture with the user application at the top and the I2C drivers are the bottom. All the layers will be independent from each other.
- ➔ All the tasks are to log important information into a log file supplied by the user in command line in a timely manner.
- ➔ The sensor data is gathered every 100 msec. The BIST task runs minor tests on the system to check if other threads have been created and attempts to read identification register in the sensors to see if it gets the expected value and check the status of the log file and log enqueue and dequeue function to see if it works. If the status of the system is positive, the Green led connected to the BBG is on. Else it sends the error log to the log file and error message to the console as well as RED led connected to BBG is on.
- ➔ The unit tests are a separate application which works on top of the entire project. We will use cmoka for this.

Task Descriptions & Basic API overview :

1. **Main Task:** Spawns all the below threads. All except the BIST threads are blocking, as soon as the BIST thread has completed its testing all the threads are unblocked. Checks if all threads are spawned and receives heartbeat in fixed intervals from all threads. In case of any error, logs to the logger and displays it on the console and turns the BBG led red. It makes extensive use of pthread library and semaphores. It sets a timer for a few seconds after spawning the BIST thread giving the BIST thread time to complete all its start up test. Only if the BIST tests are a success, it moves forward with pthread_join API.
2. **Temperature Task:** It takes the temperature data from the sensor in a periodic manner. It has several APIs to do this. The one mentioned in the software diagram is getTemp() API.

It will rely on two underlying functions called `updateTemp_protected()` and `readTemp_protected()`. These two functions have mutex locks in them so that no two process can read and write the global float variable for temperature at the same time. This will prevent contention. The basic `getTemp()` API takes the requested unit as a parameter and the internal switch case operation chooses the unit into which it is to be converted. Besides this it also has various smaller APIs in the file called `TempSensor.c`. These include: `read_config_reg()` & `write_config_reg()` /*set/read conversation rate, em operation, fault bits, sensor resolution and so on*/ `read_Tlow_reg()`, `read_Tlow_reg()` and `write_ptr_reg()`.

3. **Light Task:** It takes light data in lumens converts it into LUX every regular interval. As shown in the software diagram, it has 2 major APIs and several minor APIs. The major APIs are `getLight_8BIT()` and `getLight_16BIT()`. These 2 interfaces wraps all the internal read and write APIs. As in the case of temperature task, there are possibilities of contention hence we define two functions, `updateLight_protected()` and `readLight_protected()` so that there can be no simultaneous read and write operations. The (heartbeat) periodically. There are several smaller APIs in the file `LightSensor.c`. These include: `write_cmd_reg()`, `read_ctrl_reg()`, `write_ctrl_reg()`, `write_timing_reg()`, `read_timing_reg()`, `read_irc()`, `write_irc()`, `read_identity_reg()`, `read_intr_threshold_reg()`, `write_intr_threshold_reg()`, `read_luxdata0_adc()`, `read_luxdata1_adc()`.
4. **Socket Task:** This is a server task which remains always on waiting for a client to connect to it and request data from it. There are 3 user option as per the software diagram: `RequestTemp()`, `RequestLight()` and `RequestBoth()`. All the 3 APIs internally call the `getTemp()`, `getLight()` or both `getTemp()` and `getLight()` one after the other and sends the data back to the client to be displayed on the console. The get APIs as mentioned above use the protected APIs to avoid race condition. The request from the client is random and has a GUI which various configuration options, including the option to change the units of the data. It remains in a continuous loop unless terminated by the user. The server does not support multiple clients at the same time. Both the server and the client maintains the same structure called `payload_struct_t` which has elements like module ID, enum `Request Type` and enum `Temp_unit`.
5. **Logger Task:** The logger thread makes use of the **Posix Message queue IPC** to enqueue and dequeue the logs in the functions `Log_t loggerEnqueue(..)` and `Log_t loggerDequeue(..)`. The logs are printed in a manner that Timestamp. PID then TID then Module ID followed by log level (enum of log levels) and finally message are printed in the log file supplied by the user in command line. It requires a struct called `Log_t`, it has members as mentioned above. All the log functions will be in a file called `logger.c`. The `logger.h` will be called in all the modules that require logging. As soon as any event occurs, the log struct is filled up and the log is sent to the queue. The logger dequeues the queue in a fixed time interval.
6. **BIST task:** BIST task is the task that runs before the other threads start doing their work. In my project, I am going to make sure that only after BIST test are successful, the actual program starts running. There are several APIs mentioned in the software diagram. The `testTempSensor()` and `testLightSensor()` they read the identification register of respective

sensors. The thread status, `testThread_stat()`, can be checked using a volatile global Boolean array variable where all elements are initialized to 0 or `not_running`. Then as soon as the threads have started they will update this Boolean array element of their respective index to 1. If all indices are 1, then all threads are up and running. Finally in `testLogger_stat()` checks if the command line log file can be opened and if a dummy log message can be printed in it. As soon as all the BIST tests are completed and successful, the BIST thread exits after setting a `BIST_success` flag for the logger. As the logger sees this flag set, it realizes that all the operations were successful and prints the log for BIST. If the operations are not a success, the BIST does not exit and logs the message to the log file. It also displays it on the console and sets the led red connected to BBG.

Other modules:

1. **Timer Module:** To achieve precise timings we use posix timers for this purpose. A timer Interrupt every 100 msec calls the call back function which gives the semaphore for the respective sensor to get data from the registers.
2. **Error Handling module:** This sets and clears general purpose gpio pins connected to the respective LEDs to set red led in case of error and green in case of normal.
3. **Individual Sensor Modules:** Contains for the respective API functions for the sensors.
4. **I2C device drivers :** 2 sensors use the same I2C bus, hence proper protection and bus arbitration is used to avoid bus contention. Includes functions like `I2C_temp_read(..)`, `I2C_light_read(..)`, `I2C_temp_write(..)`, `I2C_light_write(..)` etc.

Development Plan:

	Things to do	Date
1.	Initial Software diagram and documentation	3/03/19
2.	Basic pthread frame work	5/03/19
3.	Main task functionality (start/shutdown/heartbeat mechanism)	8/03/19
4.	Hardware Interfacing completion	10/03/19
5.	Temperature task functionality	15/03/19
6.	Light Sensor task functionality	20/03/19
7.	Heartbeat reaction/recovery	22/03/19
8.	Logger task functionality	24/03/19
9.	Remote Request Socket Task	26/03/19
10.	Startup / BIST tests	28/03/19
11.	Unit tests	30/03/19
12.	Extra Credit	-----

13.	Report	31/03/19
-----	--------	----------

References:

1. AESD Project 1 pdf
2. Software diagram photos by unknown authors under the license [CC BY-NC-ND](#), [CC BY-SA-NC](#), [CC BY-SA](#), [CC BY](#)