Project 1 Report
ECEN 5013
Khalid AlAwadhi - Poorn Mehta
March 31st 2019

https://github.com/K5Ma/APES_Prj1

Project overview

In this project we created a multithreaded system on BeagleBone. This system utilizes pthreads to connect to multiple offboard sensors, which are TMP102 (temperature sensor) and APDS-9301 (light sensor). They will both share the same I2C bus. Other pthreads will be used log information and allow the board to be ready to receive commands from an external source as well as read log data from our system.

We created 5 different pThreads:

1- Main pThread: The main thread spawns the child pThreads. Any time it creates a thread it checks for errors. Once all pThreads are created, Main does checks on a global variable called AliveCheck to see what thread is alive or not. If a thread is alive, Main sends a message to the Logger to log it. In the case the Logger is dead, Main will instead log it to stdout. If any thread is not alive, main will do the same thing but in addition, it will turn an LED on. Each LED maps to a thread. USR0 LED => Logger Thread / USR1 LED => Socket Thread / USR2 LED => Temp Thread / USR3 LED => Lux Thread.
Once all threads are dead, Main will print a message to the user and blink the LEDs in a cool exit animation.

2- Logging pThread: The Logger thread is in charge of logging all messages sent between our processes. It will first create the log file to store everything into and then wait blocking until a message is sent to its POSIX queue. At runtime the user can choose the log file name and path. Any messages sent to the Logging POSIX queue will be logged alongside a timestamp, source of message and log level (which is INFO, WARNING, CRITICAL, and FATAL). If the logger is restarted using the same filename, it will delete any previous log file. Additionally, if all the threads are dead, teh Logging thread will kill itself and thus ending the program (as there is nothing to do)

3- Socket pThread: The Socket Thread runs and waits for any commands sent by an external source. Once a command is received, it will relay it to the target pThread via POSIX queues. Additionally, it will relay the targeted threads response back to the client as well as a timeout if the targeted thread does not respond.

4- Temp pThread: The Temp pThread communicates with the TMP102 sensor via I2C bus and gets timed temperature measurements (which are triggered by a signal timer set in Main). Any

measurements are sent to the Logger thread to be logged to a text file. The pthread is also able to respond to external requests such as changing the temperature data format coming in from the Socket pThread (which is coming from a Client).

5- Lux pThread: The Lux pThread communicates with the APDS-9301 light sensor and also take timed LUX measurements. Since two sensors are using the same I2C bus, we made sure I2C bus accesses are atomic. Like the temperature measurement pthread, it will also send measurements to the Logging thread to be logged to a text file. Additionally, it supports command coming in from the Socket thread (which is coming from a Client). Any changes in state will also be logged.

Also, Main sets up signal handlers so if USR1 or USR2 signals were passed, they will signal to all threads to cleanly terminate and exit. They all do that starting with the Sensor threads, then Socket will exit, then Logging and lastly Main. Gracefully and in an organized fashion they will all exit.

Our API calls:

We created a lot of API calls in our project, below is a description of each (for more information like details about each parameter please look at the actual header files on GitHub, including it will make this report much longer):

Header file: My_Time.h
Functions:

- double GetCurrentTime(): This function will simply get the current time and return its double value. It was created to simplify our code since we need to get the current time many times and having it in this format helps keeps things organized.

Header file: POSIX_Qs.h
Functions:

- void SendToThreadQ(uint8_t Src, uint8_t Dst, char* Log, char* Message): This function will send a message to a chosen pThread based on the parameters. In addition, it has extensive error handling which really helped as as we were building our project.
- Log_error(uint8_t Src, char* Err_Msg, int errnum, uint8_t Mode): This function will output UNIX errors alongside a message to either: send them to the logging thread, or just output to stdout, or both. This was created using a thread-safe error retrieval function.

Header file: LoggingThread.h
Functions:

- void LogFile_Init(char* LogFilePath): This function will be called initially when the Logging Thread is first initialized.

- void LogFile_Log(char* LogFilePath, MsgStruct* Message): This function will log messages received by the Logging Thread. It will decode the message and specify the destination and source it came from and the log level and log it to a file.

Header file: SocketThread.h
Functions:

- uint8_t SocketThread_Init(void): This function will be called initially when the Socket Thread is first initialized.
- uint8_t kill_socket_init(void): When called, this function will kill the Socket Thread.

Header file: TempThread.h
Functions:

- uint8_t TempThread_Init(void): This function will be called initially when the Temp Thread is first initialized.
- uint8_t custom_temp_reg_write(uint8_t r_addr, uint16_t r_val): Function to write to any internal register of Temperature Sensor.
- uint8_t custom_temp_reg_read(uint8_t r_addr, uint8_t *r_val): Function to read from any internal register of Temperature Sensor.
- uint8_t custom_set_temp_thresholds(void): Function to set temperature thresholds provided by defines in the header file.
- uint8_t custom_test_temp_config(void): Function to test all settings of Temperature Sensor through Configuration Register
- uint8_t get_temp(float *t_data): Function to read temperature in deg C.
- uint8_t custom_temp_init(void): Function to initialize temperature sensor

Header file: LuxThread.h
Functions:

- uint8_t LuxThread_Init(void): This function will be called initially when the Lux Thread is first initialized.
- uint8_t custom_lux_reg_write(uint8_t r_addr, uint8_t r_val): Function to write to any internal register of Light Sensor.
- uint8_t custom_lux_reg_read(uint8_t r_addr, uint8_t *r_val): Function to read from any internal register of Light Sensor
- uint8_t custom_test_lux_config(void): Function to test all settings of Temperature Sensor through manipulation of various registers.
- uint8_t custom_lux_init(void): Function to initialize light sensor.
- uint8_t get_lux(float *l_data): Function to read light level in lumens.

Start-up Tests:

Every time we run our code on the BeagleBone, it will perform start-up checks. The most important thread is the Logging thread, so that will run first and as we reach critical points we will print it to stdout (as we have nowhere to log it yet) like the thread was created successfully or the logging file was successfully created (or not). We have included extensive error checks in all threads.
The threads that spawn after the Logging thread will share similar error checks but instead of printing to stdout, they will send success or error messages via POSIX queues. to the Logging thread which will handle logging these messages to a file.

The Temp and Lux threads have even more rigorous error checks as they have off-board sensors that we must ensure work. Here is what will be done at start-up for each:
Temp Sensor:

- Test powering on sensor
- Test Temperature Sensor reset
- Test setting THigh at 30 deg C
- Test setting TLow at 20 deg C
- Test that setting temperature thresholds was successful
- Test default Temp Config check
- Test Fault Bits
- Test Extended Mode Set & Clear
- Test Conversion Rate

After all these tests, then the Temp thread will be allowed to operate normally.

Lux Sensor:

- Test powering on sensor
- Test Gain and Integration Time
- Test Interrupt Control Register
- Test Interrupt Threshold TLow
- Test Interrupt Threshold THigh
- Test ID Register

Like the Temp thread, after all these tests the Lux thread will be allowed to operate normally.

Unit Tests:

For our unit tests, we decided to do 5:
1- Log file path: The purpose of this test is to see if we can choose a file destination which will store all log messages when executing the program. The default file name is LogFile.txt which will be saved in the current program's directory.

2- Thread Creation: The purpose of this test is to see if we can create pThreads successfully and then join them before exiting the Main pThread.

3- Test Message logging between threads: The purpose of this test is to send a message from Main to the Logging thread and have it be logged to a file. This will test our IPC and message structure. As well as our error handling.

4- Temp sensor start-up tests: The purpose of this test is to test our start-up operation of the Temp sensor. It does all the tests described in the start-up tests section.

5- Lux sensor start-up tests: The purpose of this test is to test our start-up operation of the Lux sensor. It does all the tests described in the start-up tests section.

Block Diagram