Advanced Embedded System Development

BeagleBone Linux System Design Project 1 (110 Pts)
Due Dates:

Architecture Document - No Later Than Sunday, March 3rd (Midnight) – earlier is encouraged
Code Submission and Final Report - Sunday, March 31nd (Midnight)
Demo – To be scheduled after Spring Break by Doodle.

Revision 2019.2.17

1 Overview

In this project your team will be designing and implementing a "smart" environment monitoring device using the BeagleBone Green development board along with two offboard sensors. An example of the sensor's application would be a refrigerator where the purpose is to monitor for out-of-bounds conditions such as an exterior door left open via a light sensor or the internal storage temperature rising too high for properly preserving food.

The project will be managed and evaluated in 3 development phases:

1) Architecture and Review
2) BeagleBone Development and repository submissions
3) Demonstration and Final Report

You will apply numerous topics discussed in class and build on previous assignments to create your own kernel, and application software for the target BeagleBone Green Development Board. The design will include concurrent software concepts for Linux that will interact with both User-Space and Kernel-Space in addition to multiple connected devices.

Conceptually, the application is to concurrently monitor two sensors (temperature and light sensors) connected to the same I2C bus and log data along with exceptional conditions to a single file on the system. The sensors should be monitored periodically. Additionally, an external interfacing task will service requests from an off-system (or simulated) host inquiring the system's status/logs.

Please read this complete document before beginning work.

2 Learning Objectives

After completing this assignment, you will be able to:

- Write a multithreaded C-program using pThreads in User space to collect data from device drivers
- Interface with two externally connected I2C sensors.
- Design a synchronized buffer to collect logs from other threads
- Design threads to operate in an event loop

3 Guidelines

Projects can be performed in teams of 1-2 people. All coding MUST adhere to the C-programming style guidelines posted on Canvas. Failure to do so will result in point deductions. As with any commercial project, you'll be expected to generate documentation that shall be submitted along with your code to your git repository. This can be in the form of commented code, design notes, readme support, pdfs or other files types.

The exception to this is your System and Software Architecture document, that should be independently submitted to Canvas by the associated due date so that it can be reviewed.

Additionally, for the Final Report you will need to generate and include a code dump report. This should be a single file in a .pdf format that you will submit to the Canvas Project 1 Assignment page. It should be generated from the git repository code with the annotated tag **project-1-rel.** This allows a check for plagiarism. No other formats will be accepted. As with all course work, any online sources must be cited in the code files and any preexisting library code must retain licensing/credits in the code. You should comment your own code indicating the author. Failure to give credit is a violation of the Honor Code

To generate the code dump report, "cd" to your top level directory, and execute this one liner command to copy all files and folder contents into a single file to submit.
$ find . \(-name '*.c' -o -name '*.h' \) -exec cat {} \; > allcodefiles.txt

4 Resources

These resources may be helpful in your development:

BeagleBone Green

- http://beagleboard.org/green
- http://beagleboard.org/buildroot
- https://github.com/beagleboard/buildroot
- https://bootlin.com/doc/training/buildroot/buildroot-labs.pdf
- Lots more at bootlin.com…

Texas Instruments Temperature Sensor

- http://www.ti.com/lit/ds/symlink/tmp102.pdf

Broadcom Light Sensor:

- https://www.broadcom.com/products/optical-sensors/ambient-light-photo-sensors/apds-9301

Mastering Embedded Linux Programming, 2nd Ed, Simmonds. Chapter 9 – Especially the section on I2C.

5 System and Software Architecture, Review

Like any substantial commercial embedded software development project that designed meet project and product requirements, you are to create a system and software architecture diagram along with some documentation of your design. This must be turned into the instructor team on Canvas before writing any code. It can be submitted earlier than the due date, if you wish to begin coding sooner.

The purpose of this activity is to:

1) Assimilate and understand the requirements of the project.
2) Reinforce the practice of thinking before coding a complex software project to:
    a. Guide/inform a development plan – e.g. Identify high risk areas to address first
    b. Identify and address where complexity may exist in your design

Your architecture submission should include the following items:

- A diagram of what the software design and the interactions between hardware and major software components
    - Indicate tasks, modules, kernel, software communication interfaces, etc.
    - Existing libraries and design elements planning be used in your design and explicitly included in your compilation.
        - For example, in user space you should be using pthread for your task design
    - Indicate the hardware interfaces for sensors and external communications
- Describe each task names, software structures, and responsibilities.
- A first-cut definition of the needed API/functions for tasks and major functions
    - Indicate the type of interface and functional support needed. E.g. IPC types, get_temp() and get_light() API into your tasks to retrieve data.
- Other important concepts will be what is user vs. kernel space software. Indicate if something is a kernel module or a system call. etc.

Note: You should note on your figure and documentation where you intend use existing library code, drivers, etc. and what will be your own code.

Submit your architecture diagram(s) and documentation on Canvas. In addition, it would be useful (but not absolutely required) if you met with any member of the instruction team during office hours, by special appointment or through online means to discuss your design before you start programming. Present your assumptions and come with questions. Small adjustments in assumptions and answering questions early can be a time saver and have big leverage later in the development process.

6 System and Software Development Requirements

You will need to compile and install your own Linux Kernel image for your BeagleBone Green development board. There are many ways to install your kernel image on a target board with the preferred method of installing by putting the image on your own microSD card. Netbooting is another option.

The Serial Debug cable will be very important for debugging your kernel installation and interacting with your boot process.

6.1 Kernel/OS Configuration

You will need to show that you have installed your own version of the kernel on the BeagleBone Green and that you are not using the stock image.

6.2 Device Drivers / Kernel Modules

Because both sensor devices will be connected to the same I2C bus (i2c-2), a mechanism is needed to provide exclusive access to the I2C bus as two independent tasks will be trying to read and write to two independent I2C connected sensors. This can be performed in user space (ioctl,open/close, read/write) or as a device driver wrapper written for the I2C libraries as a part of the kernel or as a kernel module themselves. Use a synchronization device such as mutex or semaphore to coordinate the two task's usage of the I2C bus.

6.3 Software Requirements and Tasks

You are to divide the monitoring and processing of the environmental monitoring application into several tasks, described further in the following sections. Starting with a Main task, a separate task is required for each of to the two sensors. (More information on these sensors can be found in the resources section.)

All tasks should also be logging "state" to a shared log file interface to a Logging task, also discussed below. Finally, a Remote Request Socket Task is required to support external queries concerning the condition of the system, sensors, etc.

The tasks should be implemented using the pthread library.

6.4 Main Task

The Main Task should use an Asynchronous Threading strategy and has a couple of responsibilities.

1. Create all the children tasks
2. Ensure that all tasks are still alive and running on some regular interval
3. Eventually cleanup your application properly if you wish to stop your task. Meaning, the Main task should be "joining" on child threads and issuing exit commands to threads.
4. Log error information to the console user and indicate an error condition (e.g. missing sensor) with the BBG USR LEDs
    a. This could be its own independent task or a part of the main task's functionality.

6.5 Temperature Sensor Task

This task interacts with the TMP102 Temperature sensor. This is an I2C connected sensor with a resolution of 0.0625 Degrees Celsius. It is suggested to write a single read and single write function as the interface to this sensor for all registers needed. For this sensor you will need to be able to perform the following distinct operations:

- Write the Pointer Register
- Read Sensor Tlow register
- Read Sensor Thigh register
- Read Sensor Temperature data register
- Read/Write the Configuration Register
    - Configure the sensor to go in and out of shutdown mode
    - Configure the sensor resolution
    - Read the Fault bits

    - Set/Read the EM operation
    - Set/Read Conversion Rate

Temperature data should be able to be read and converted from its binary form into its proper decimal representations. The temperature sensor must report correct temperature (including negative temperatures).

This task should be able to respond to external requests for temperature data via IPC commands using an IPC mechanism of your choice (Queues, Pipes, Sockets, etc.). This task should be able to report numerous temperature formats including Celsius, Kelvin, and Fahrenheit if requested in different API calls.

Additionally, this task should use timers to wake up every so often to gather new data from the externally connected sensor. This timeout should occur at a reasonable amount of time (A time greater than 1 msec).

[**Extra Credit +3**] Manipulate the Tlow & Thigh registers along with the fault bits and alert pin to show temperature moving outside of your provided range. Use an LED or some type of connected lab tool to show the TMP102 module alert pin changing.

6.6 Light Sensor Task

You will need to design a task to interact with a APDS-9301 Light sensor which can sense both visible light and Infrared light. It is suggested that you create a set of command message routines to read/write all registers for 8-bit and 16-bit modes, so you have a single defined interface in your software that wraps your interface reads and writes. This task will need to perform the following operations.

- Write to the command register
- Read/Write the Control Register
- Read/Write the Timing register
    - Set the Integration time
    - Set Gain
- Enable and disable the Interrupt Control Register
- Read the Identification Register (Think startup tests!!!)
- Read/Write to the Interrupt threshold registers
- Read sensor LUX data using the ADC registers (Data0 and Data1)

The Lumen output can be determined by reading the LUX data ADC and using conversion equations in the datasheet. Use these data sheets so you can report the actual luminosity of the sensor. You also should be able to read the two different types of light corresponding to visible light and IR light (Channel 0 and Channel 1). Just like the temperature sensor, this task should periodically sample the LUX data, and support an API from other tasks to request the recent LUX data from each sensor, along with a “state” whether it is currently “light” or “dark”. You can choose the appropriate light/dark threshold values. Additionally, when this task senses a change in state, light to dark or dark to light, it should send a log message that there was a change.

[**Extra Credit +3**] if you can manipulate either/both the light sensor and temperature sensors interrupt and alert pins to connect to the BBG and have an interrupt fire and be recognized for the temperature/light swing.

6.7 Synchronized Logger Task

This task will be dedicated to accepting logs from various on-board sources. It must be able to connect to multiple targets as well as receives logs from many different sources (threads/processes). This interface will need to have protection from multiple log sources. How this is done is up to you. Either a log queue, an API message, or via control synchronization.

This logger should write data to a log file which file path and file name should be configurable at runtime. Meaning, you can pass in different file/paths at run time using either command line options or configuration files. Only the logger can write to the log files. Each time the logger is restarted and is provided the same log file name, it should delete the previous file and start fresh or backup the previous file and start a new one.

This task should gracefully close the file if this process is forced to close or requested to close. When requested to do this, it should do its best to flush its log queue, close file handles and disable any connections with external tasks.
Each log entry must contain the following information

- Timestamp
- Log Level
- Logger Source ID (your choice/naming convention)
- Log Message – Application specific

Logs entries should be able to contain both character strings and integer/float data so that tasks are able to send different types of data via the log task interface. For example, the Temperature Sensor task should be able to report temperature data to this logger while a Light Sensor task can light/dark alerts.

Logs should be produced at reasonable intervals from all the tasks. Meaning when certain events occur (initialization, data conversion, system failures, etc.).

6.8 Remote Request Socket Task

This task's responsibility is to accept a socket request from another process/task or the network and make API calls to your sensor tasks for data. This task is a good example of a way to test your application while its "live" by requesting/injecting API commands into the internal messaging structure. Most importantly, this socket will allow external programs to make requests for the sensor data.

You will need to create a second process or use an off-system host that can create a socket connection with your Remote Request Socket Task to show that you can make remote requests for your sensor data and potentially other API commands.

6.9 Other required design elements

6.9.1 Message APIs

As discussed above, the design must implement some type of inter-thread messaging system for tasks to interact with one another. This implementation is up to you. For example, each task can have its own message queue, or there can be a single message queue and tasks that distributes event flags to each task directly, or you can use shared memory or utilize sockets.

A simple set of messages should be implemented. At a minimum, the set should support the following:

- Heartbeat notification from all threads to the main task
    - Either as a request-response from Main task to individual tasks as a “ping”, or a periodic message from each task to the Main task with Main monitoring (in some manner) the presence/absence of the report.
- Startup tests Initialization Complete Notifications (Success/Failure) from each task
- Error Message reporting
- Sensor Tasks’ Data Requests (temperature/light)
- Log Messages
- Requests to close threads from main to other tasks

As mentioned above, each task should have a heartbeat API message that can be sent to Main to report its current state. If heartbeats are not received, then appropriate actions should be taken to either recover or fail gracefully. If you want to add more functionality to this list, you certainly can.

6.9.2 Startup/Built-In Self Tests (BIST)

Before you have your application begin steady state operation monitoring temperature, light, and logging errors, tasks should have some tests run at startup to verify that hardware and software is in working order. These tests should include:

- Communication with the Temperature sensor that you can confirm that I2C works and that the hardware is functioning
- Communication with the Light sensor that you can confirm that I2C works and that the hardware is functioning
- Communication to the threads to make sure they have all started and up and running.
- Log messages should be sent to the logger task when individual BIST tests have finished along with an indicator if the hardware is connected and in working order. If something does not startup correctly, an error code should be logged and the USR Led turned on.

6.9.3 Unit Tests

You must write unit test code that can be built and run on your software. This means that your software must be designed in a modular or layered way so that software can be tested when not fully integrated. Suggestions for unit tests include, but are not limited to

- Inter-thread communication. For example, something testing Synchronized Buffer/Queue
- Logger
- Temperature Sensor conversions with mocked data
- Light Sensor conversions with mocked data

You are free to suggest and implement the type of unit tests and the application of those unit tests. However, you will be required to unit test at least 3 different pieces of software (at the module level). This code should be checked into your git repository along with your application. Be sure to the organize and name unit test code files in coherent manner.

7 Project 1 Deliverables and Rubric

7.1 General Project Requirements and Limitation

During the project development there are multiple deliverables over time. They include

1) System and Software Architecture document
2) BeagleBone Development and Repo submissions
3) Demonstration and Final Report

During the BeagleBone development phase you must regularly commit your code to a git repository as described in the following section.
A breakdown of the points for the project are below:

Item	Value (points)	Due Date
System and Software Architecture Package and Git repo link to Canvas	5	March 3rd@ midnight or sooner.
Consistent use of Version Control	5	Regularly
Adherence to documented Coding Style Guidelines	5	
Kernel customization & build	3	
Main task functionality (start/shutdown/heartbeat mechanism)	10	
Light Sensor task functionality	10	
Temperature task functionality	10	
I2C bus device access protection mechanism	2	
Heartbeat reaction/recovery	5	
d Logging task functionality with sane shutdown	5	
Remote Request Socket Task	10	
Startup / BIST tests	5	
Unit tests	5	
Final Report, Code Dump, Repo Tag for final code	15	March 31nd @ midnight
Demonstration, GDB use and Practical Interview	15	To be scheduled after Final Report
Extra Credit Temperature Alerting	3 (extra	

	credit)	
Sensor Interrupt driven functionality	3 (extra credit)	
Kernel module I2C sensor device access protection	10 (extra credit)	

7.2 Version Control and Platform Support

The project code must be must be submitted via a git repository. All commits must be made before the final date. Please add your group members and project link to the initial assignment submission.

Any online sources must be cited in the code files and any preexisting library code must retain licensing/credits in the code. You should comment your own code indicating the author. Failure to give credit is a violation of the Honor Code.

If you created a private repository, you need to add the instructor team as a collaborator to the project.

You should be able to demonstrate integrated use of version control. That is, code should not be committed for your full project in 1-2 commits. Instead, it should be submitted in smaller commit blocks consisting of new features or bug fixes. There will be no minimum number of commits required, but there should numerous commits from both members. You must support multiple files with the proper scope of those files. Meaning, main.c cannot include all of the code. Points will be deducted if a reasonable attempt is not made to structure the code in a system-wise logical fashion.

Some examples of how to split up the project into multiple independent features can include:

- Adding file structure (directories and files for API/IPC methods, tests, etc), descriptions, copyrights, etc.
- Design stubs (C-programming): These types of commits represents your software architecture design or your module outline. This can include all defined prototypes, commented function descriptions, and empty function definitions. No function implementations. You will need a few of these as there are multiple modules to code.
- Feature Commits: These commits represent your actual feature developments.
    - BIST
    - Unit test and completed feature code
    - Different Tasks
    - Communication interfaces
- Partner Integration Commits: These commits would represent necessary changes need to integrate different features from partners.
- Bug Fixes: These commits should represent any bugs that you found and fixed.
- You will need to place a git annotated tag with the name **project-1-rel** where you want the project to be graded. This tag MUST be placed before the due date of the project. For reference see: https://git-scm.com/book/en/v2/Git-Basics-Tagging

7.3 Project Demonstration and Final Report

The final phase of the project requires a demonstration and practical interview on the assignment. All individuals will need to meet individually with the instructor team, not as a group. An updated System and Software Architecture diagram should be presented that reflects the final design of your project.

The interview will be a mixture of student demonstration, technical questions from the interview team, and a question about how you might address architecturally adding a new feature. Topics can include, but are not limited to:

- Overall project functionality validation – i.e. met/missed project requirements
- Synchronization issues and implementation
- Accuracy of sensors
    - Test of ability to sense light and lack of light
    - Accurate sensing of temperatures including below 0 degrees Celsius
- Unit Test Validation Demonstration
- Github Repository questions
- Use of GDB