ECEN 5013-002

PROJECT 1 REPORT

By Shreya Chakraborty

31st March 2019

INDEX

1. Project Requirements

2. Hardware Connection Diagram

3. Software Design Diagram

4. Project Description

5. Installations

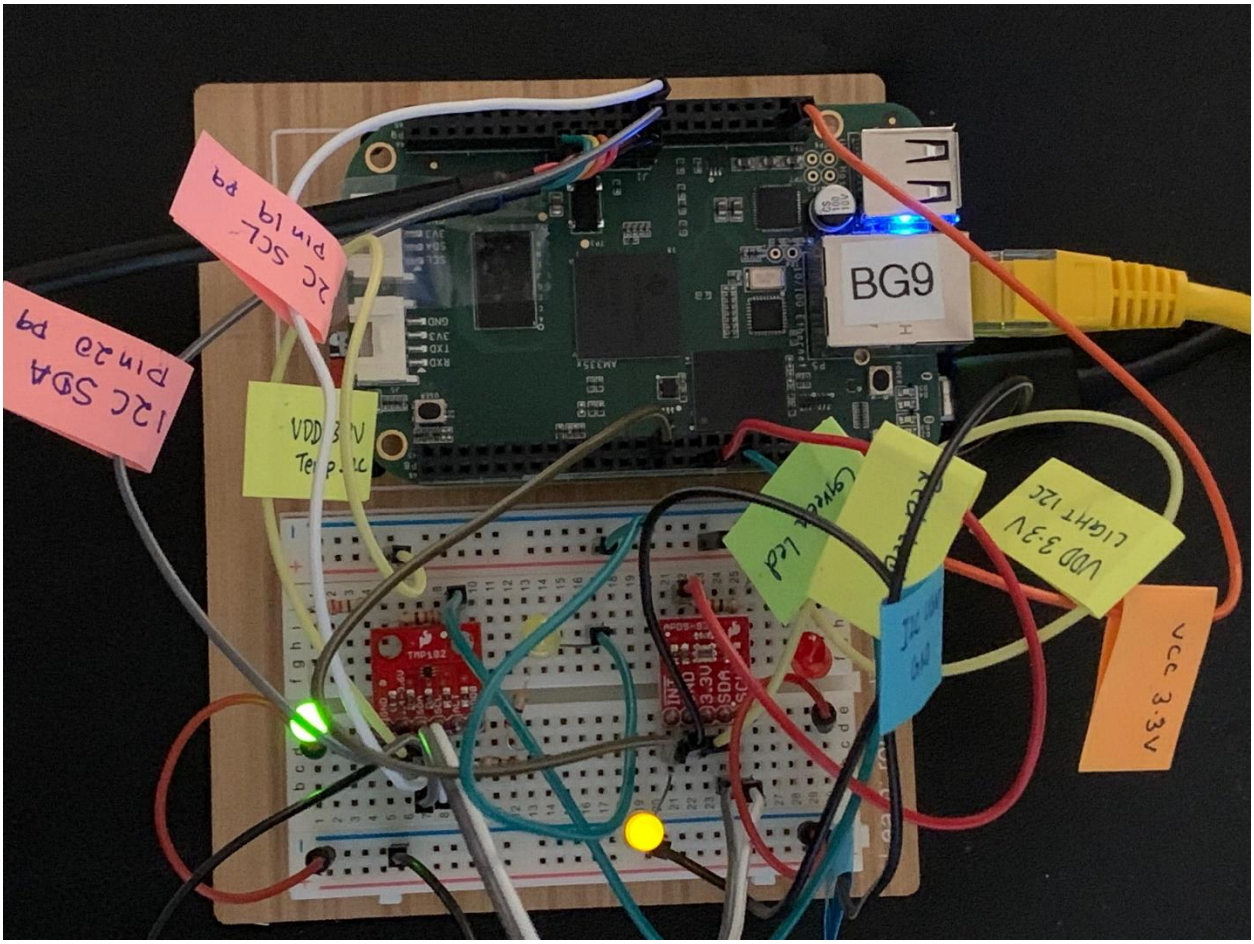
6. References

PROJECT REQUIREMENTS

1. Project 1 requires Beaglebone Green 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.
2. The relation between each task and connection to various modules are as given in the
   software architecture diagram. All boxes having black colored arrow are threads spawned by the main. The various modules in each task are colored coded and so are the arrows.
3. All the tasks periodically send a heartbeat message to the main task saying that they are alive.
4. The Socket task must service requests from the external client. The requests include get temperature in unit of our choice and so on.
5. The temperature task and the Light task must be able to perform distinct register read and write operations. The list of all the operations are provided in the link.
6. Some form of IPC must be used by all tasks to communicate with the logger task. And logger task alone will write to the file given in the command line argument.
7. The logger task will enqueue logs received from all other task modules and dequeue it and write it to the file.
8. The built in self test task will check if all other threads are spawned, sensors are connected and functions, I2C initialization is successful and logger queue working status and closes. Only after the successful completion of BIST the main program can start.
9. 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.
10. The sensor data is gathered every 1 sec. The BIST task runs minor tests on the system before the start of the main program, if successful only then the actual program starts.
11. In case of error, the error message is displayed on the console along with the red led on. In normal working scenario, the red led is off and green led is on. Green led refers to system ok while red led refers to error.
12. The unit tests are a separate application which works on top of the entire project. The tests must be written to check the Sensor APIs and logger. We will use cmocka testing framework for this.

HARDWARE CONNECTION DIAGRAM

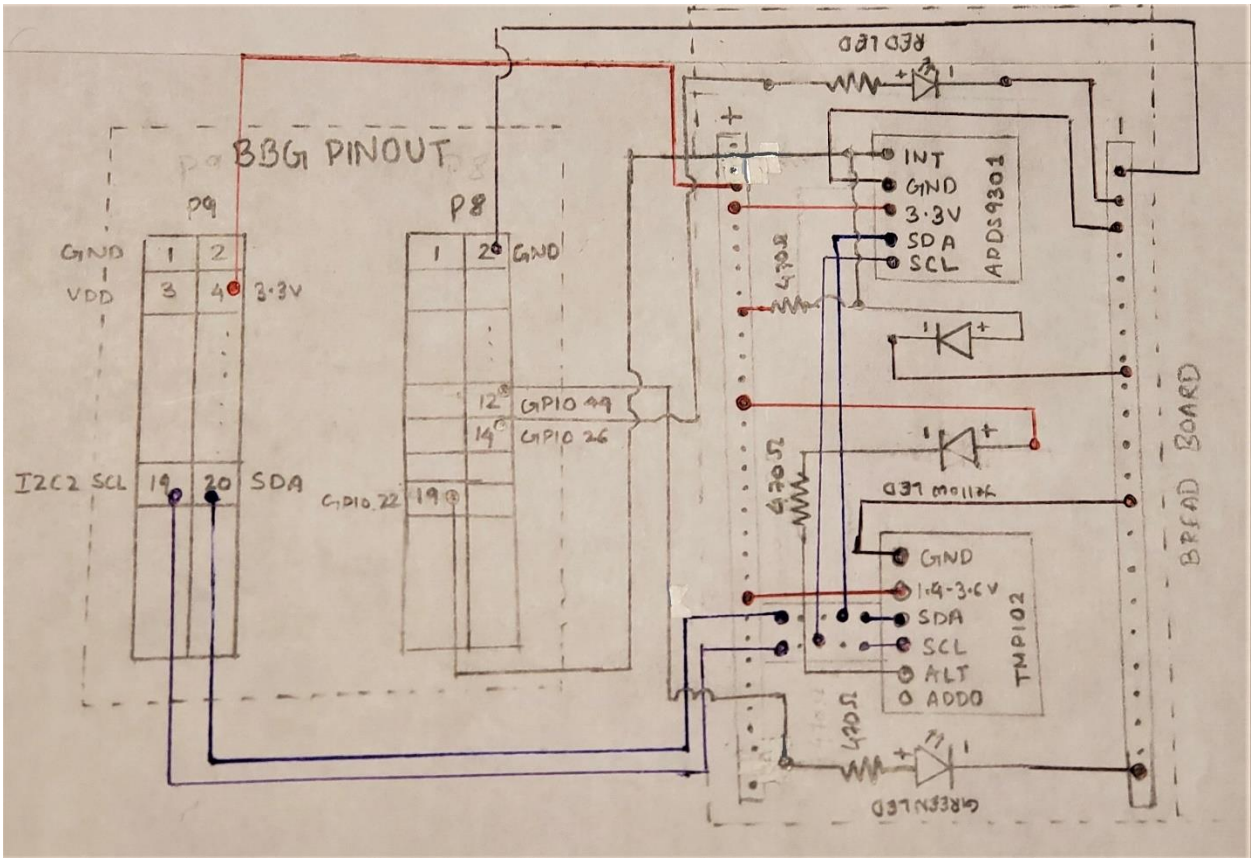
How it looks…



Items Required for set up:

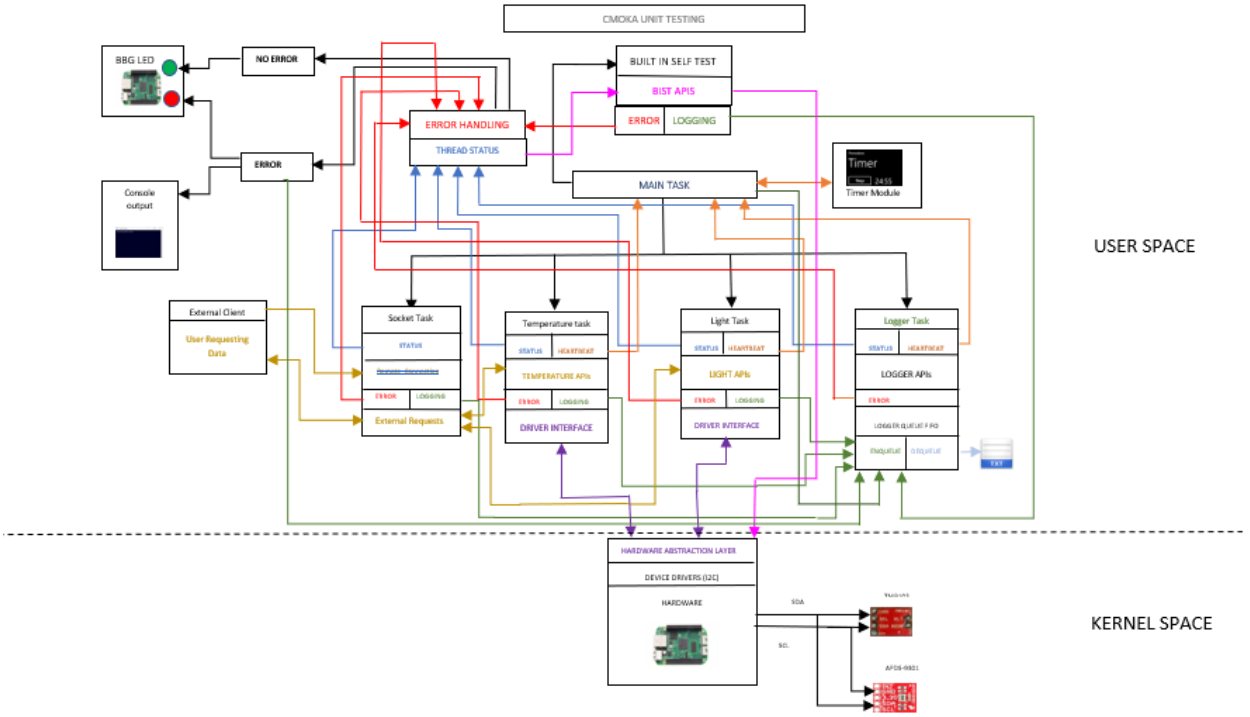
1. Beaglebone Green
2. FTDI cable
3. Ethernet cable
4. TMP102 sparkfun
5. APDS9301 sparkfun
6. 4 LEDs
7. 470 ohm registers ideally. (I used 2 220 ohms in series as I did not have 470)
8. Bread board and bread board base plate.
9. Bunch of male to male jumper wires
10. Sticky notes for labelling the wires (always a good practice!)

What it actually is…



1. The SCL and SDA on BBG is on P9_19 and P9_20 respectively. They are connected to the SCL and SDA of the sensors.
2. The P8_2 GND is connected to the common GND of bread board. The P9_4 VCC is connected to the common VCC of breadboard.
3. The cathode of the GREEN LED used for heartbeat is connected to common GND and the anode is connected to GPIO_44 or P8_12 via a resistance of 470 ohms in series.
4. The cathode of the RED LED is connected to the common GND and the anode is connected to GPIO_26 or P8_14 via a resistance of 470 ohms in series.
5. There are two YELLOW LEDS used. One for temperature threshold and other for light threshold. The lower one in the above picture(YELLOW LED 1) is used for temperature threshold and the upper one (YELLOW LED 2) is used to light threshold.
6. The cathode of the YELLOW LED 1 is connected to the ALT pin of the TMP102 via a resistance of 470 ohms in series. The anode is connected to the common VCC.
7. The cathode of YELLOW LED 2 is connected to the common GND. The anode is connected to the INT pin of APDS9301. The INT pin is pulled up using a pull up resistor of 470 ohms to common VCC.

SOFTWARE DESIGN DIAGRAM



Main parts of software design:

1. Main task spawns all the threads, checks heartbeat from light, temp and logger thread and ensures clean exit. Uses pthread APIs extensively.
2. BIST task performs the built in self check. It is a thread spawned by the main after it has spawned all the other threads. It performs the tests while the other tasks wait for semaphore from the BIST task. If all tests are successful, it posts semaphores for the test of the waiting threads and exits. Otherwise it turns on the RED LED for error and clean exits the program. The main functionality will not start unless the bist tests are successful.
3. Light task gathers light data from the APDS9301 sensor every 1 sec using a posix timer. It also sends light data in response to the external client request. It can also change the threshold values of sensor based on the input threshold values by the external client. Based on the upper limit of the threshold, it logs a message saying it day or night. The YELLOW LED 2 is connected to the INT pin of the sensor and remains on if the lux value is below the threshold and blinks periodically if the lux value exceeds the threshold.
4. Temperature task gathers temp data from the TMP102 sensor every 1 sec using a posix timer. It also sends temp data in response to the external client request. It can also change the threshold values of sensor based on the input threshold values by the external client. Based on the upper limit of the threshold, it logs a message saying threshold exceeded and lights up the YELLOW LED 1.

5. Logger task gathers logs from all the tasks. The tasks use posix message queues for interprocess communication. The tasks enqueue the logs and the logger thread dequeues the logs and prints it in the log file.
6. Socket task accepts connection request from the remote clients and services the requests. Depending on the requests it calls APIs pertaining to specific tasks.
7. External client has a bunch of options for the user to test the program. It sends the request over socket and the socket task sends response.
8. Timer has been used for temp and light task with a call back function that posts the semaphore for the threads to get the sensor values.
9. Heartbeat module checks the global flag whose corresponding bist are set by all threads except bist and socket thread. If it finds one of the bits clear, the declares the thread corresponding to the bit position as dead. The red led is turned on as well. If all is proper then the GREEN LED is toggled based on the frequency of heartbeat.
10. On death of a thread, the program does clean exit by using flags to break out of while loops of the all the threads and flushes the log.
11. MRAA library for i2c is used for communication with the sensors.
12. CMOCKA is used for testing the APIs.

The entire project is organized into different files in the Project1 folder of my repo for convenience.The src contains all the source files , the include files are in the includes folder, the cmocka tests are in the test folder. Configurations are in config folder The doxygen files generated are in doxygen folder. The entire project has a common makefile with various targets.

To make the project for BBG do the following:

1. “make build” – to make the project and send the files over to the BBG home/Project1/code & output & config folder. The output folder contains executables.
2. “make test” – to make all the test files execute them and send them over to BBG.
3. “make doxygen” – to generate doxygen report
4. “make ssh” – to open a terminal with the ssh.
5. “make all” – to do all the above at once.
6. “make clean” – to clean out the executables.

To run the project on the BBG do the following:

1. For the main program: “./project.out log.log 3” on the BBG whose usage is : “./project.out [log file] [log level]”
2. To run the external client : “./client_test.out” on another terminal of BBG.

PROJECT DESCRIPTION

Here I will try to explain the general working and the flow of the project as best as I can:

1. In the start the main task spawns all the threads expect the bist thread. The thread status flag is set if the threads have been spawned successfully. Then the main spawns the bist thread and joins it. Meanwhile other threads are waiting on a semaphore to start. The bist thread performs test to check the light sensor by reading its ID reg "**Test_LightSensor**()", temperature sensor by reading the temp register "**Test_TempSensor**()", it checks the thread status flags to check if all the other threads have spawned properly "**Test_AllThreads**()" , perform a test for I2C init to see that I2C bus initialization is successful "**Test_I2C**()" and checks the logger queue functionality "**Test_loggerQ**()". If bist checks are successful, it sets the "**BIST_allOk**" flag and bist thread exits.The **checkBistResult**() function called in main which checks the "BIST_allOk flag" if okay it posts the semaphores for all the other threads to execute "**PostBistOkResult**()" and starts the heartbeat check "**startHeartbeatCheck**()". In case the bist tests are unsuccessful, the bist flag will be 0 , the checkbist function fails the RED LED is turned on, the threads can't get started and hence exit and the program exits cleanly.
2. As soon as the Temp task is spawned,it waits on the semaphore from the bist task. As soon as it gets the semaphore it logs the corresponding message, initializes the timer and starts it. In its while loop it waits on a second semaphore from the timer module. The timer call back functions is called every 1 sec and it calls a function "**giveSemSensor**()" which posts the semaphore for the timer and then it can perform "**readAndUpdateTemp**()" The read and update function calls the "**TMP102_getTemperature**()" API which basically performs I2C read of the temperature register. The value of temperature received is put into a global static variable. It also checks if the value of temperature threshold has been changed, if changed then it performs the "**TMP102_setTempThreshold**()" function to set the new values of threshold received from the external client into the threshold registers. The temperature exceeding the threshold is checked by checking the AL bit of the config register. We also call "**set_heartbeatFlag**()" in the while loop which sets the bit corresponding to the module ID of the task. The module ID of temperature task is 2 in my case.
3. The similar operation occurs in the case of light task with the little change in function names. It calls "**readAndUpdateLight**()" which calls "**APDS9301_getlight**()". It also does similar checking for threshold, in addition it also checks the INT pin of the sensor "**APDS9301_checkINTPIN**()" to see if there is any interrupt. The interrupt is generated when the light does above the threshold value.
4. Similar to the above threads even logger checks for bist, waits for a semaphore and performs the dequeuing in the while loop. In logger task, I have a "**logger_queue_init**()" which does a mq_open(). The "**LOG_ENQUEUE**()" function checks the descriptor, fills the log struct and enqueues the log using mq_send internally. The while loop perfoms me_receive and dequeues the logger queue and prints it to the file. There are several macro functions described for various level of log like log debug, log info, log warn and log error.

The 2nd command line argument takes in the log level. The level 2 refers that log level 2, log level 1 and log level 0 will be shown on the console. The most important is log error, followed by log warning and log info.

5. The socket task does not send a heartbeat to the main. Its blocking on the **accept**() . It checks for the semaphore from bist task and in its while loop waits for a client to connect. It calls several specialized APIs that it calls to full fill the demands the external client like **GET_TEMP_KELVIN**() for temperature in kelvin, **kill_temp_thread**() to kill the temperature thread , "**RemoteThresholdValues**()" to send the new threshold values for the temperature task.
6. The heartbeat module has two main APIs "**set_heartbeatFlag**()" and **startHeartbeatCheck**(), the set function is called in each thread which sets the corresponding bit in the flag depending on the module ID number. The start cheak calls a timer with the call back function "**heartbeat_timer_callback**()". This functions checks the flag to see which module did not send a heartbeat, if all are received, it makes the flag zero and sends the notification that's all threads are working. If one of the tasks is missing then it itendifies the task based on the bit field number and sends the notification and performs the **SystemExit**() and disables the heartbeat timer. A point to be noted is that there is a separate timer for both heartbeat and the sensor values.
7. For error handling, in the led module, I am doing gpio manipulation to on and off the leds. The GREEN LED toggles based on the frequency of the heartbeat which is 2 secs. The RED LED is turned on only when there is an error. If the sensor is disconnected then I get a notification based on the return value of the sensor API. And it works fine if the sensor is reconnected again.
8. For synchronization between threads, semaphores have been used extensively. For protection of the global variable operations mutexes have been used. There are two sensors using the same I2C use but to ensure that no two sensors read or write at the same time a combination of semaphore and mutexes have been used.
9. The temperature and light register read and write APIs are in the files tempSensor.c and lightSensor.c. The underlying I2C drivers used have been referenced from the mraa I2C library.
10. The remote client is continuously on and gives the user a list of options to choose from. Depending on the option, it sends the socket task an indication and the socket tasks calls the corresponding function the client has requested.
11. For exiting the program, the signal can be send by the client or the user can input CTRL-C from the keyboard. It internally calls the SystemExit() which basically sets all the thread kill flags which causes the threads to stop executing in the while loop and exit.

INSTALLATIONS

1. **Installation of Mraa host Cross Compile for the libmraa library used in myI2C.c and myI2C.h**

For dependencies:
sudo apt-get install build-essential autoconf libtool cmake pkg-config git python-dev swig3.0 libpcre3-dev nodejs-dev
sudo apt-get install gcc-arm-linux-gnueabihf g++-arm-linux-gnueabihf

1: git clone https://github.com/intel-iot-devkit/mraa.git
2: cd mraa
3: mkdir host_crossbuild
4: cd host_crossbuild
5: vim mraa_arm_cross.cmake

Then change the cmake file as follows:

```
##start of file
  SET(CMAKE_SYSTEM_NAME Linux)
  SET(CMAKE_SYSTEM_VERSION 1)
# specify the cross compiler
SET(CMAKE_C_COMPILER /home/shreya1809/buildroot/output/host/usr/bin/arm-linux-gcc)
 SET(CMAKE_CXX_COMPILER /home/shreya1809/buildroot/output/host/usr/bin/arm-linux-g++)
  # where is the target environment
SET(CMAKE_FIND_ROOT_PATH /home/shreya1809/buildroot/output/host/usr/arm-buildroot-linux-      gnueabihf)
  # search for programs in the build host directories
  SET(CMAKE_FIND_ROOT_PATH_MODE_PROGRAM NEVER)
# for libraries and headers in the target directories
SET(CMAKE_FIND_ROOT_PATH_MODE_LIBRARY ONLY)
SET(CMAKE_FIND_ROOT_PATH_MODE_INCLUDE ONLY)
SET(SYSTEM_USR_DIR /home/shreya1809/buildroot/output/host/usr/arm-buildroot-linux-gnueabihf)
##end of file
```

6: cmake .. -DBUILDSWIG=NO -DBUILDSWIGNODE=NO -DBUILDSWIGPYTHON=NO - DCMAKE_TOOLCHAIN_FILE=mraa_arm-linux-gnueabihf.cmake
7: sudo make
8: sudo make install

Scp libmraa.so ,libmraa.so.2 , libmraa.so.2.0.0 from the host_crossbuild/src folder to the /lib folder on BBG.

2. **Installation of Cmocka**

Copy the cmocka.c and all the cmocka header files in the project directory.

Include that while compiling the project

3. **Installation doxygen**
   sudo apt-get install doxygen
   doxygen -g doxygen.config
   make changes in the doxygen.config file.

REFERENCES

1. https://www.96boards.org/blog/cross-compile-files-x86-linux-to-96boards/
2. https://iotdk.intel.com/docs/master/mraa/i2c_8h.html
3. https://github.com/intel-iot-devkit/mraa/blob/master/examples/c/gpio.c
4. https://github.com/sparkfun/SparkFun_APDS9301_Library/blob/master/src/Sparkfun_APDS9301_Library.h
5. https://github.com/torvalds/linux/blob/master/drivers/hwmon/tmp102.c
6. https://www.geeksforgeeks.org/socket-programming-cc/