PROJECT 2 SOFTWARE ARCHITECTURE

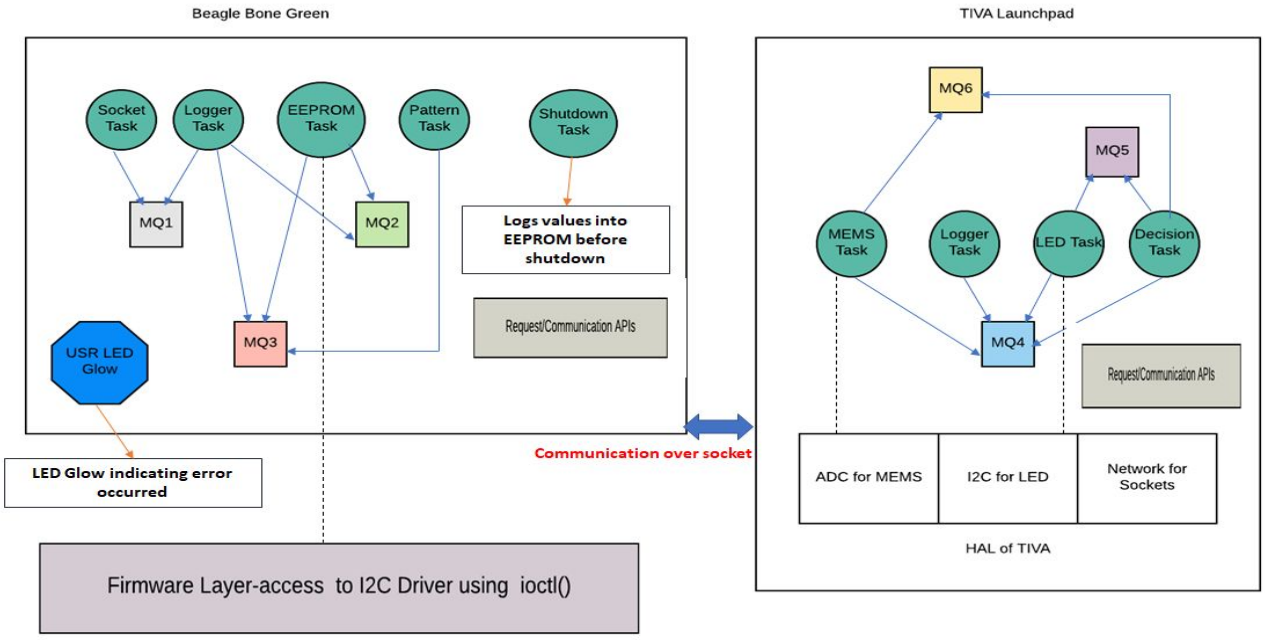
Bhallaji Venkatesan, Divya Sampath Kumar

Interactive Noise Monitoring System

INTRODUCTION

The project aims at monitoring and displaying the noise level in the surrounding region using a MEMS Audio Sensor (ADC Interface) and an 8x8 LED Matrix (I2C Interface). We intend to monitor the ambient noise through the MEMS sensor and display a noise profile periodically on the I2C based LED Matrix. The two devices would be connected to the TIVA TM4C1294XL with the remote logger server running on BeagleBone and the logger client running on TIVA with various API supports for the TIVA-Beaglebone interface. Data Collection and display would be carried out in the TIVA end and operations such as logging of sound intensity values and area characterization as “noisy” or “quiet” depending on a specific threshold would be done on the BeagleBone Green. In addition to this, we plan to have the LED Matrix controlled over sockets to reconstruct historical noise patterns. For a robust client-server model, we intend to implement a NVM component to cater power loss solutions on the Beaglebone Green end.

SOFTWARE ARCHITECTURE



MQ1 - Socket Task Log
MQ2 - EEPROM Task Log
MQ3 - Pattern Task access to Logger Task and EEPROM Task Logs
MQ4 - Common Logger Queue for MEMS, LED and Decision Task
MQ5 - Queue for LED Task to act according to Decision Task
MQ6 – Queue MEMS Task to act according to Decision Task

TASKS INVOLVED

BBG Tasks:

1. Socket Task - Caters to incoming requests from the TIVA Launchpad
2. EEPROM Task - To store values send by TIVA into the EEPROM
3. Remote Logger Task - Performs data and error logs into a log file and EEPROM MQ
4. Shutdown Task - Logs values into the EEPROM Device before shutting down
5. Pattern Task - Recognizes patterns (period of silence, high noise, varying intensities in the audio logs)

TIVA Tasks:

1. MEMS Audio Task (MAT) - Reads values from the MEMS Sensor and logs values into Logger task
2. LED Display Task - Display LED Patterns based on the surrounding noise curve
3. Logger Task - Logs data from other tasks into the Log Queue
4. Decision Task - Interacts with MAT, takes decision on when to display LED Patterns

DOCUMENTATION ON NEEDED STRUCTURES, TASK NAMES AND TASK RESPONSIBILITIES

TIVA End: (Server)

Task related APIs:

1. xTaskCreate - To create the required tasks
2. vTaskPrioritySet() - To set the priority for the tasks
3. vTaskDelay() - To delay a task execution during required conditions
4. vTaskDelayUntil() - To delay a task execution during required conditions until a particular event
5. vTaskResume() - To resume a task that was paused or stopped
6. vTaskStartScheduler() - To start the scheduler to schedule the tasks
7. vTaskEndScheduler() - To stop the scheduler to end the tasks
8. xTaskNotifyAndQuery() - To Notify and query the tasks

HAL Layer APIs (In process of finalizing):

ADC Related APIs-

1. struct adc* adc_init(uint32_t index, uint8_t bits, uint32_t hz)
2. int32_t adc_start(struct adc * adc)
3. int32_t adc_stop(struct adc * adc)

I2C Related APIs (using the FreeRTOS+ IO API) :

1. BaseType_t FreeRTOS_ioctl(Peripheral_Descriptor_t const xPeripheral, uint32_t ulRequest, void *pvValue);

Messaging IPC APIs:

1. xQueueCreate() - To create a message queue for IPC
2. xQueueSendtoBack() - To send message onto the message queue to another task
3. xQueueReceive() - To receive the sent message on the message queue.

Sockets related APIs:

1. FreeRTOS_socket - To create a TCP/UDP socket
2. FreeRTOS_bind - To bind to a local port number
3. FreeRTOS_connect - To Connect TCP socket to a remote socket
4. FreeRTOS_listen - To listen for incoming connection requests from remote sockets
5. FreeRTOS_accept - To accept a connection on a TCP Sockets
6. FreeRTOS_send - To Send data to a TCP socket
7. FreeRTOS_sendto -To Send data to a UDP Socket
8. FreeRTOS_recv - Receive data from a TCP socket
9. FreeRTOS_recvfrom - Receive data from an UDP Socket
10. FreeRTOS_setsockopt - To Set socket options
11. FreeRTOS_closesocket - To close the corresponding socket

BBG End: (Client)

Pthread APIs:

1. Thread Usage
2. Synchronization Devices

IPC APIs:

1. Message queue APIs

Firmware APIs:

1. IOCTL syscall

Socket APIs

Used Structures and associated Enums

Various Log levels associated with the log packet:

```
typedef enum loglevel_t{
LOG_DATA_VALID,
LOG_DATA_REQ,
LOG_CRITICAL,
LOG_INIT,
LOG_ERR
}loglevel;
```

Various Task Sources for log packet:

```
typedef enum srcid_t{
SRC_BB_MAIN,
SRC_BB_SOC,
SRC_BB_LOGGER,
SRC_BB_EEPROM,
SRC_BB_PATTERN,
SRC_BB_SHTDWN,
SRC_TIVA_MAIN,
SRC_TIVA_MEMS,
SRC_TIVA_LED,
SRC_TIVA_LOGGER,
SRC_TIVA_DECISION,
SRC_DEFAULT,
}srcid;
```

Log Packet Structure:

```
typedef struct log_packet_t{
struct timeval *time; //Timestamps use time.h
loglevel level; //Log Levels based on criticality of logs
srcid sourceid; //Source of Logs
char logmsg[256]; //Message to be logged
int32_t crc; //CRC check for error checking mechanism
}logpacket;
```