ECEN 5823 – Course Project Final Report
4/28/2020
Brian Ibeling

Subsystem Overview

For this individual contribution, the entire system should be taken into consideration. Please view this information under the "Project Overview" section of the Team Proposal.

In summary, the BLE Motion Detection system seeks to detect motion via a BLE Server device and capture an image and/or alert a user of the event via a BLE Client device.

Subsystem High Level Design

For this individual contribution, the entire system should be taken into consideration. Please view this information under the "High Level Design" section of the Team Proposal.

Project Sensor(s) and Hardware

The BLE Motion Detector will utilize a PIR sensor to detect when motion has occurred, which will trigger an event to occur on a client BLE device. If additional funding and time were available, a camera would also have been added to the BLE Client. For the purpose of this Proof of Concept design, an LED will be used to simulate a camera, and used as an alert to a user in a separate location.

This system will utilize the SEN-13285 PIR motion sensor from SparkFun. The SEN-13285 PIR motion sensor will be connected to a BLE Server (running on a Silicon Labs Blue Gecko device) via an analog sense line (GPIO input), as well as a 5V and GND connection to power the device. Details for how to connect to the sensor are provided here: https://learn.sparkfun.com/tutorials/pir-motion-sensor-hookup-guide. Using the Blue Gecko (BG) User Guide (https://www.silabs.com/documents/public/user-guides/ug279-brd4104a-user-guide.pdf), GPIO on pin PA2 on the EXP header will be used to sense/detect motion. Figure 1 below shows an example of the GPIOs available on the BG device. Figure 2 below shows the system block diagram with the physical connections for the PIR Sensor and the specific LED to be used provided at the top.

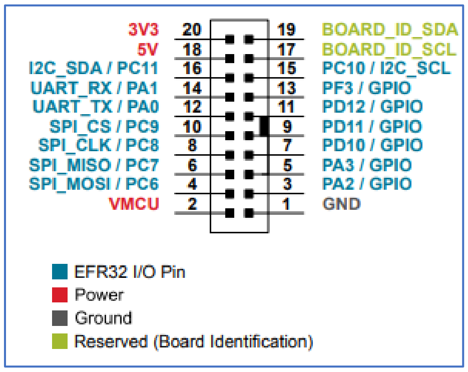

Figure 1: Blue Gecko EXP Header

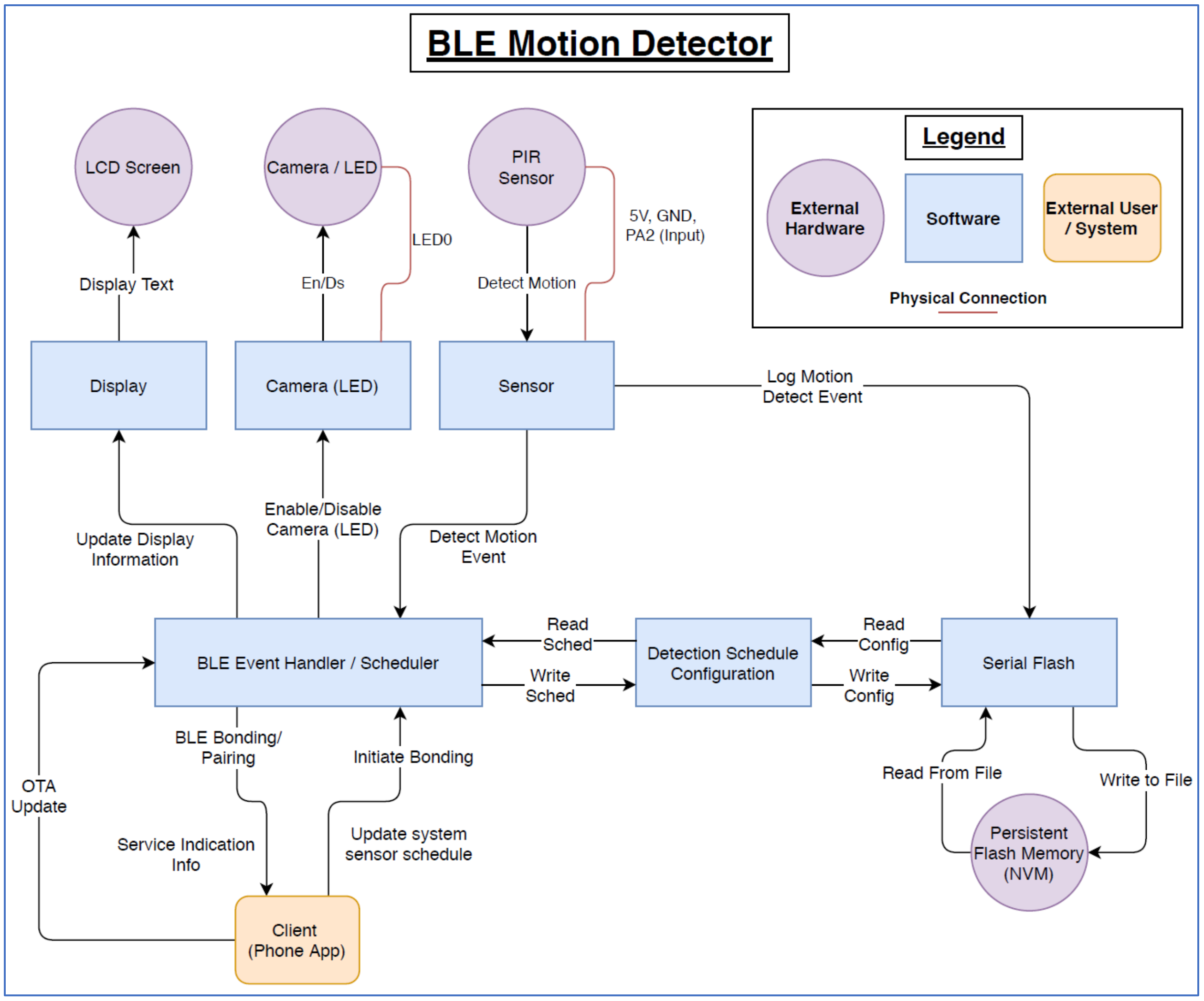

Figure 2: BLE Motion Detect System Diagram

BLE Command Table

The Bluetooth software server device and client device command table is provided at the following link: https://drive.google.com/open?id=1JuBjCE5ndGXVqu2AbVrdDHeW3Tyw-_Ss. This is a link to the excel spreadsheet uploaded to the BLEMotionDetector Google Drive folder for this project to a file titled "Course_Project_BG_Server_Command_Table.xlsx".

Services

Two services will be exposed once the server has a bonded connection to a client. The first will be a motion detected service, which will update and provide indications to the bonded client to the timestamp of the last motion event captured, as well as the sensed range of motion detected. The second service will be to monitor persistent memory and log data, providing data on the number of log events captured and the percent full on-board memory is.

Memory/Logging

The Memory and Logging subsystem will handle read/write events with on-board persistent flash memory. This will also data to be retrieved after power cycle and power loss events. This will also include the deletion of data or information no longer needed and specified by the end-user. Two primary pieces of information will be stored in flash memory – logging data and the user set sensor sample schedule. Logging data will be focused on each motion detect event, with the timestamp of the event and the approximate range of motion captured. The sensor sample schedule will be a user-specified schedule for which sensor sampling should occur. This will allow the user to specify when they wish to monitor and be alerted of detected motion events.

User Interface/Display

The display subsystem will manage information displayed the BLE Blue Gecko server LCD screen. These will showcase the current BLE connection state of the server with a client (disconnected, connecting, bonding, bonded, advertising). Additionally, the current motion detected state will be display as well, showing if motion has been detected in the last 1 second interval and the approximate distance of the motion detected.

Low Power Design

The system will utilize a scheduler and events driven by the BLE command/event stack to ensure the system is responsive while also ensuring the lowest energy usage possible. Events from external hardware (PIR sensor, button press) will be handled by interrupts to ensure minimal resources are used to handle the event (avoid polling) and the system can be in sleep mode as much as possible.

Figure 3 below shows the Blue Gecko and the peripherals available within each energy mode. Given the use of only GPIOs for system components, Energy Mode 2 (EM2) should be achievable for a majority of the system's operational life. Inputs are handled via GPIOs (EM4 functional) with external interrupts triggering the start of new events cycles (EM3 functionality), with the Low Energy Timer (EM2 functionality) being needed for driving system events . Read/write events with Serial Flash will require EM1, as well as BLE radio events. The goal will be to keep these events at a minimum to maximum energy savings.

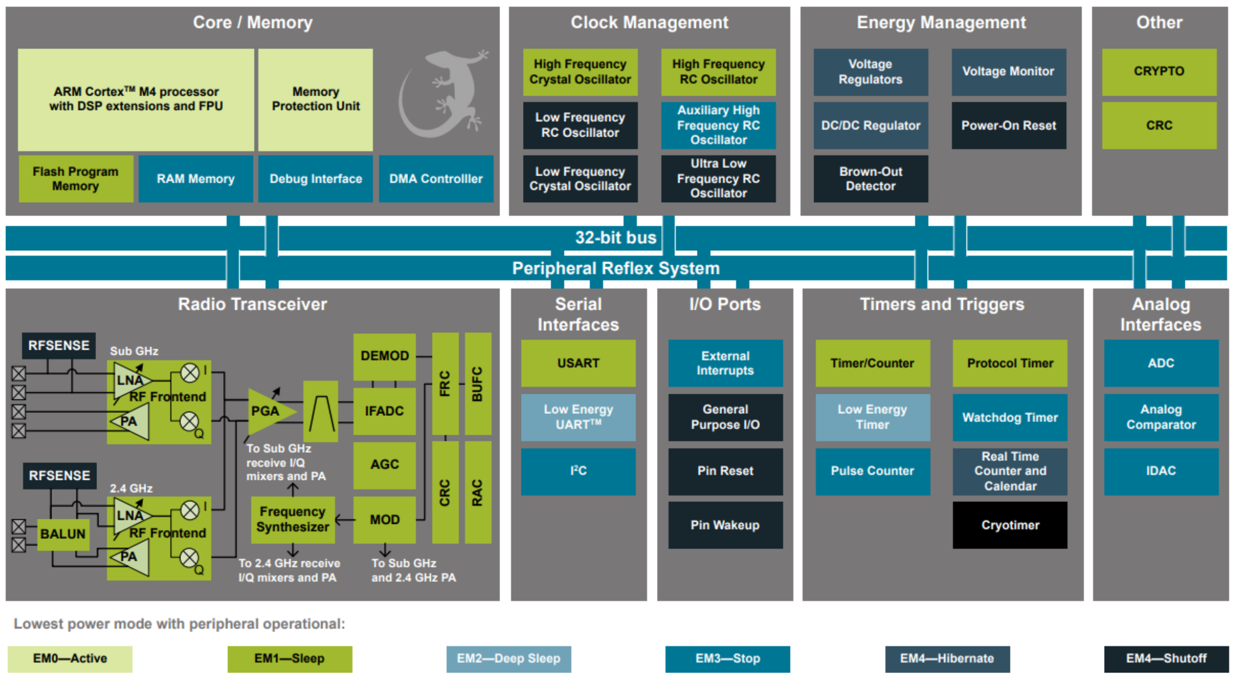

Figure 3: Blue Gecko Energy Modes

Below details the various events expected to be handled by the BLE Motion Detection system and the energy mode needed to complete each task.

BLE Device Pairing

1. System Boot (EM0)
2. BLE Server advertising (EM1)
3. Client device discovers and initiates bonding
4. Server begins bonding and pairing process (EM1)
5. Bonding complete, awaiting events and enter deep sleep mode (EM2)

LCD Update (1Hz Rate)

1. System booted, in sleep mode (EM2)
2. BLE soft_timer event triggers at 1Hz rate (EM2)
3. LCD Display updated with latest data (EM2)

Motion Detect Event

1. System booted, in sleep mode (EM2)
2. Motion detect event triggers PA2 GPIO to be set to active low (EM2)
3. Interrupt triggered via GPIO (EM2)
4. LED0 (camera) enabled (EM2)
5. Timestamp and event logged to Serial Flash (EM1)
6. After 3 seconds, LED0 (camera) disabled (EM2)

Update Detection Schedule

1. System booted, in sleep mode (EM2)
2. Update of Detection Schedule triggered by pressing PB0 – interrupt triggered (EM2)

3. User specifies schedule to set based on hour range selected (EM2)
4. User saves Detection Schedule (EM1)

OTA Update

1. System booted, in sleep mode (EM2)
2. User initiates OTA via Client device (smartphone app)
3. Server receives requests and receives OTA data (TBD energy mode)
4. OTA update written to serial flash NVM (EM1)

The energy usage for the system was validated using the Simplicity Studio Energy Profiler. It was discovered that updating the LCD at a 1Hz rate uses significant energy, and is shown in Figure 4 below. It should be noted that while idle, average energy usage is ~3 uA while in EM3, and ~5mA when updating the LCD. For comparison, Figure 5 shows the idle system energy usage with the LCD updates disabled at ~4.6uA. The small increases in energy usage in Figure 5 are when the motion detection event has occurred and data is written to flash memory, we see current usage increase to ~500uA, as shown in Figure 6. In the event a user wishes to save energy and only view information via the connect client application.

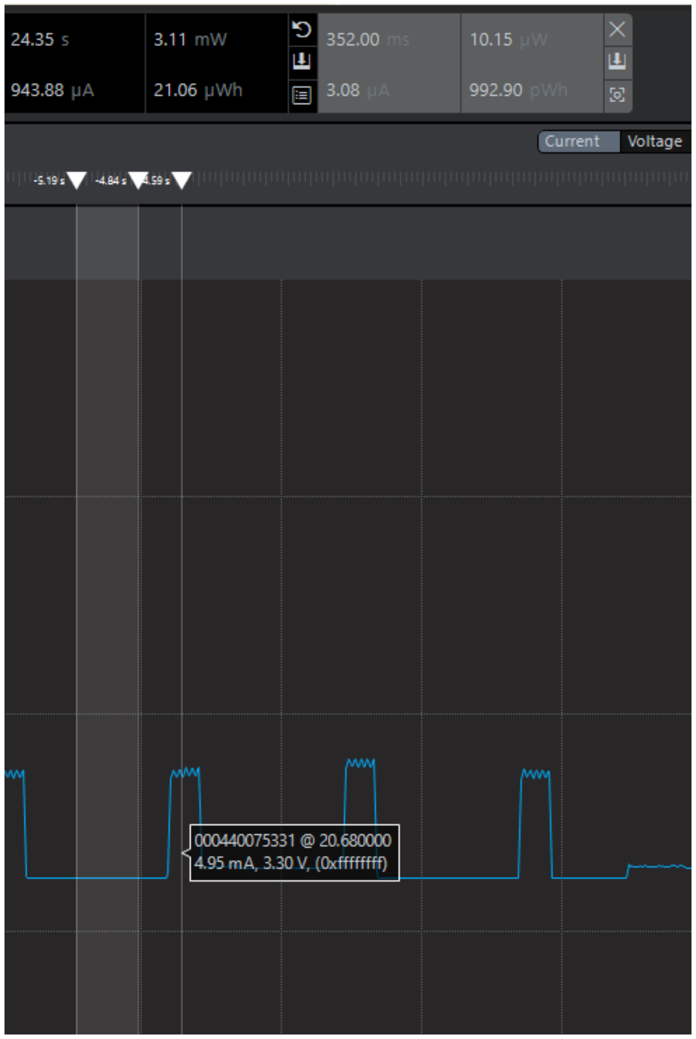

Figure 4: Energy Usage with LCD Enabled, Updating at 1Hz rate

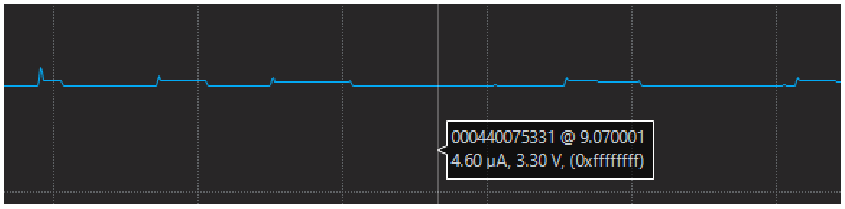

Figure 5: Energy Usgae without LCD Updates at 1Hz

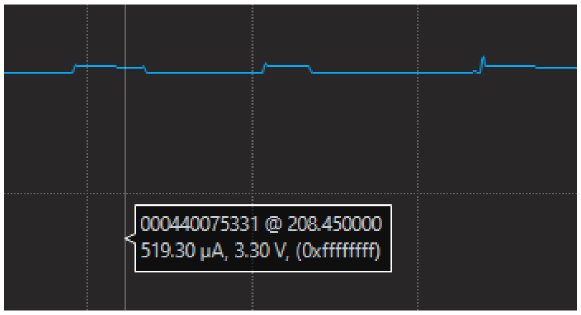

Figure 6: Motion Detect events and writes to flash memory

To additionally help illustrate the low energy usage of the system, Figure 7 below shows the energy usage while the device is bonded and the config detection schedule is updated. The large block at the beginning shows the bonding/pairing event, with current usage around 5mA. Otherwise, energy usage has peaks around 500-600 uA.

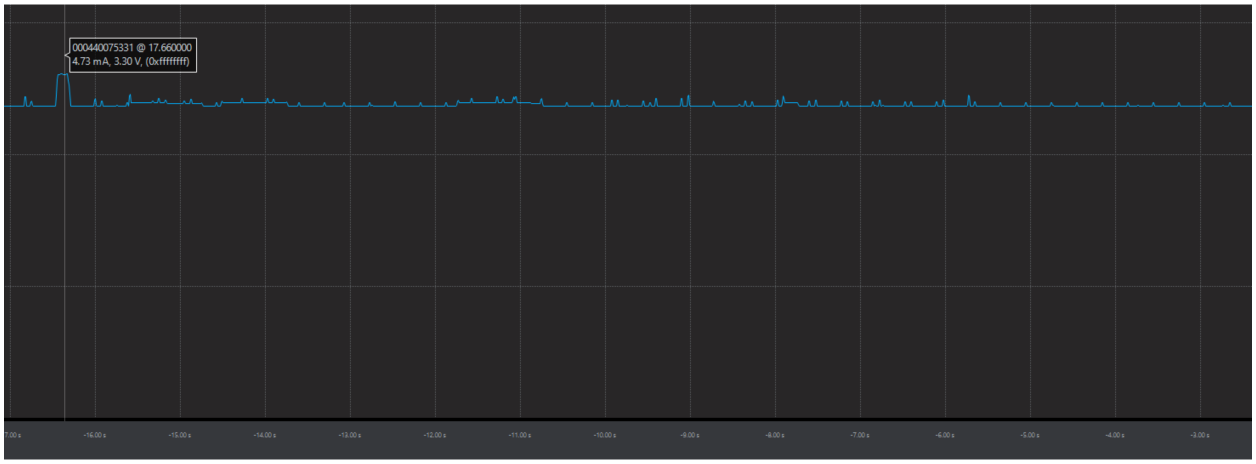

Figure 7: Energy Usage during Client Bonding and Normal Operation

Project Schedule (Updated 4/28/2020)

Table 1 below outlines the schedule of project completion.

Table 1 :BLE Motion Detect Project Schedule

Completion Date	Task Completion Dates
4/11/2020	• Outline software utilizing ECEN 5823 course homework assignments as a starting point. Ensure that BLE server able to connect to BLE client (phone app) and LCD screens updates with BLE state. (complete) • Place order for PIR sensor to be shipped from Sparkfun. (complete)
4/21/2020	• Integrate PIR sensor with system and is able to report motion detection. (complete) • Integrate PIR sensor state with LCD screen to report to user current state • Add functionality of persistent NVM flash memory (complete) • Ensure able to store recorded PIR sensor events with timestamp to flash memory. (complete)
4/26/2020	• Ensure able to write and read config schedule for reporting motion detection. (complete) • Add functionality to provide OTA firmware updates from the Phone App. (complete) • Final product integration and testing of all software and hardware. (complete)
4/28/2020	• Finalize recording of course project. (complete) • Author project report and documentation. (complete)

Project Lessons Learned

1. Updates to the LCD display is fairly high current usage, and should be disabled or updated infrequently if implementing a low-energy system.
2. BLE Service Characteristic Writes can be done with or without a response from the Server to the Client device, and supports both encrypted and unencrypted writes.
3. When uploading a new application via an OTA update, overwriting the same application only requires the application.gbl file while a new application image requires the application.gbl and the appLoader.gbl files.
4. Getting a proper time source for a BLE device can be a logical challenge. Paths for reading NTP time for this application for example could be that this is synced each time a client device is connected. Keeping proper time on network connected devices is difficult as is; doing so for BLE or other low-energy application adds another layer of complexity.
5. Having previously worked with flash memory, I didn't understand how the key applied until implemented for this project. Being able to read/write separate sections of flash based on a provided key was highly beneficial and simplified development.

Project Final Status

While some of the functionality of the project changed as the implementation evolved, the core goals set out with the project were successful. The system was able to detect motion events based on a set schedule of hours configured by a user, and the user was able to read detect event times on a client device. This windowed schedule was also able to be set from a client Smartphone device, which proved to be a much cleaner and more intuitive interface than I had initially designed (using Push Buttons to set the schedule). Read and writing serial flash was successful and helped to maintain a consistent schedule across system power cycles. Finally, OTA updates proved challenging initially to create the .gbl image files, but was an interesting and unique feature to be able to add to this project. This project proved well to expand on knowledge gained throughout the semester, and also highlighted that writing modular code up front with our homework assignments made for a much cleaner and easier to use implementation for this final project. A detailed list of all project functionality is given in the project README, found here: https://github.com/CU-ECEN-5823/final-project-assignment-ibelingb