A Detailed Breakdown of the Personal Security Device

CS199 Final Report

Karen Truong
UID: 304019042
March 2014

Introduction

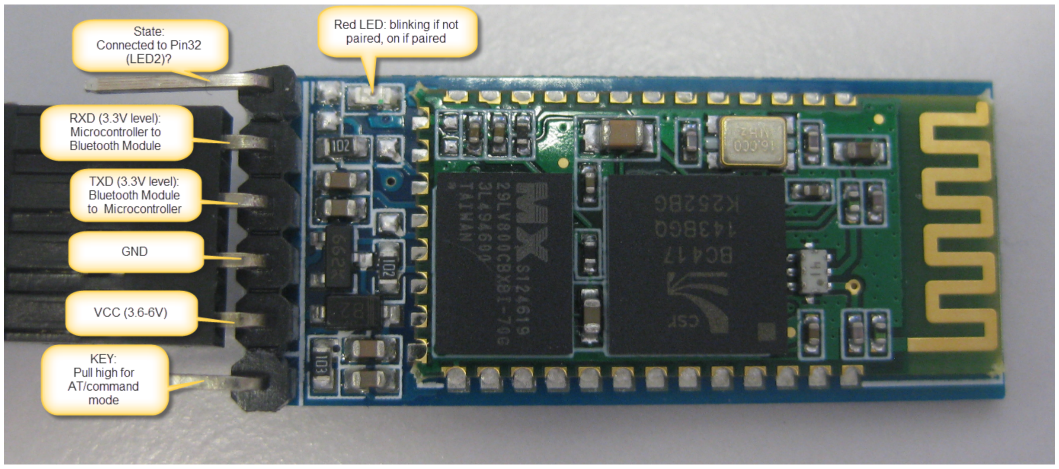
My research project this quarter was a portable, personal security device (PSD) for the Laboratory for Advanced Systems Research. The personal security device was physically assembled by Dr. Sarrafzadeh's Embedded Systems Lab. It consists of an Arduino Mega 2560 with 16 analog input pins, 54 digital I/O pins, and a USB connection to power and upload sketches [1]. In regards to memory, this Arduino has 256KB flash memory, 8KB SRAM, and 4KB EEPROM, which is an important design limitation we will consider as we develop the device. A shield is attached on top of the microcontroller to add space for other modules that were soldered on, including Bluetooth, GPS, Wi-Fi, and an inertial measurement unit (IMU). Lastly, an LCD keypad shield is attached atop the first shield for use as a display later on.

In order to support all the additional modules mounted on the Arduino shield, several new libraries had to be added. All of the supplemental libraries were found online on open-source sites and then imported into the Arduino IDE 1.0.5, an open-source application which compiles and uploads sketches to Arduino platforms. I will discuss the procedure, issues, and results I obtained when testing each module of the personal security device.

Bluetooth

The PSD includes a JY-MCU Bluetooth serial port module onto the shield to add wireless capability via serial port communication. This component of the PSD was arguably the most important functionality to verify first, as the medical devices will be communicating to the PSD and home station over Bluetooth.

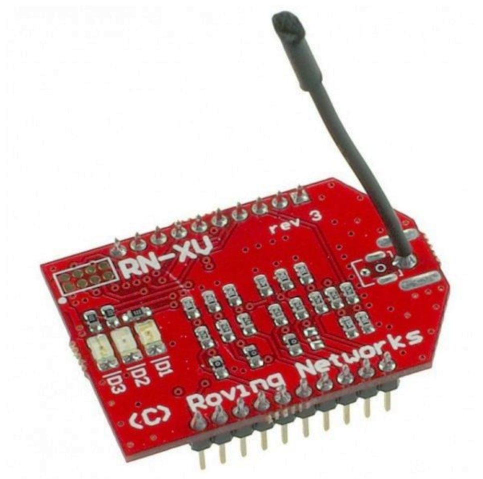
I first tested command mode of the device to verify that I could change the settings on the Bluetooth module (e.g. name, baud rate, etc.). These commands are described in detail in the Advanced User Manual from Roving Networks [7]. Keeping the device in its default slave operation state and switching it back to data mode, I used a Bluetooth dongle attached to my laptop to discover the device and establish a Bluetooth connection, which is confirmed by a solid red LED on the device. I experimented with the Bluetooth communication by using the serial monitor to transmit data to the laptop and a terminal emulation application such as PuTTY and HyperTerminal to secure a connection over the COM port corresponding to the outgoing channel of the pairing, which provides the laptop a means of sending data to the Arduino. Simple print statements on both ends verified that the data was reaching each end host correctly.


State: Connected to Pin32 (LED2)?
Red LED: blinking if not paired, on if paired
RXD (3.3V level): Microcontroller to Bluetooth Module
TXD (3.3V level): Bluetooth Module to Microcontroller
GND
VCC (3.6-6V)
KEY: Pull high for AT/command mode

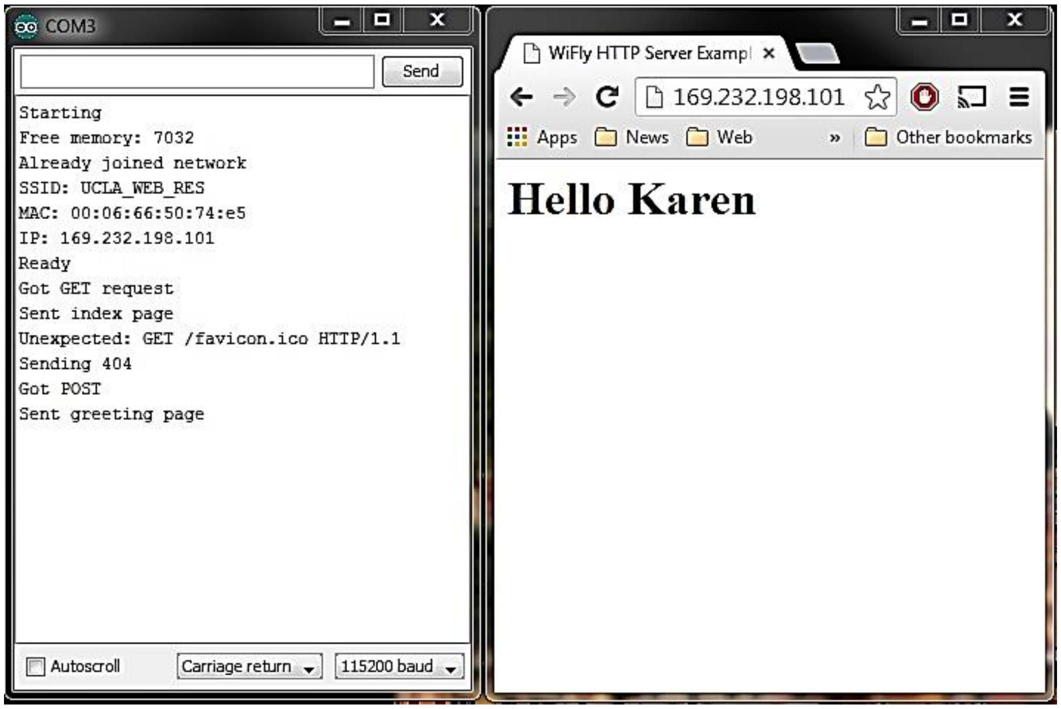
I ran into some problems with the transmissions to the Bluetooth module, particularly because its RXD and TXD pins were connected to Serial1. Data sent from my terminal to the shield was not processed correctly at Serial1, so I decided to wire the transmission and receive pins of the Bluetooth board to digital pins 10 and 11, respectively. Digital pins can be used to simulate serial ports using the SoftwareSerial library provided by the IDE. I made this decision based on previous examples found in tutorials online and the support for a BluetoothSerial class in the SoftwareSerial library. I also consulted the graduate student who put together the PSD to confirm that the Bluetooth communication from laptop to Arduino was producing garbled data over the Serial1 pins.

Wi-Fi

A Roving Networks RN-XV W-Fi module provides additional wireless support. This particular version "incorporates 802.11 b/g radio, 32 bit SPARC processor, TCP/TP stack, real-time clock, crypto accelerator, power management unit and analog sensor interface" [8]. The module has two modes: ad-hoc/command and data mode. Under ad-hoc mode, a user can use SET commands to change WLAN, IP, and hardware settings. Similar GET and status commands exist to access metadata about the module. Functionally, there are also commands like `ping` and `close` to take action at the TCP and ICMP level, as well as file I/O interface. Lastly, there are three LEDs: a red and green light to represent TCP and IP interface, and a yellow one to indicate RX/TX data transfers.



Initially, when powered, both the red and green LEDs on the RN-XV 171 will blink quickly. Once I entered command mode and set the SSID to a valid name, I saved the settings and rebooted the device which outputted information regarding MAC, IP, and gateway addresses of the network if it successfully connected. The red light disappeared while the green LED flashed less frequently. I imported the WiFlyHQ Library which abstracts some of the ad hoc commands I had manually done for the connection setup into a function. The library also provides sample programs to test TCP connectivity, as well as code to set the microcontroller up as a HTTP client or server. The figure below is the output of executing one of those example programs as a server.

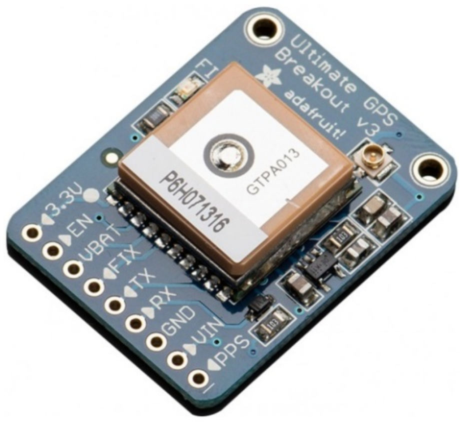


```
Starting
Free memory: 7032
Already joined network
SSID: UCLA_WEB_RES
MAC: 00:06:66:50:74:e5
IP: 169.232.198.101
Ready
Got GET request
Sent index page
Unexpected: GET /favicon.ico HTTP/1.1
Sending 404
Got POST
Sent greeting page
```

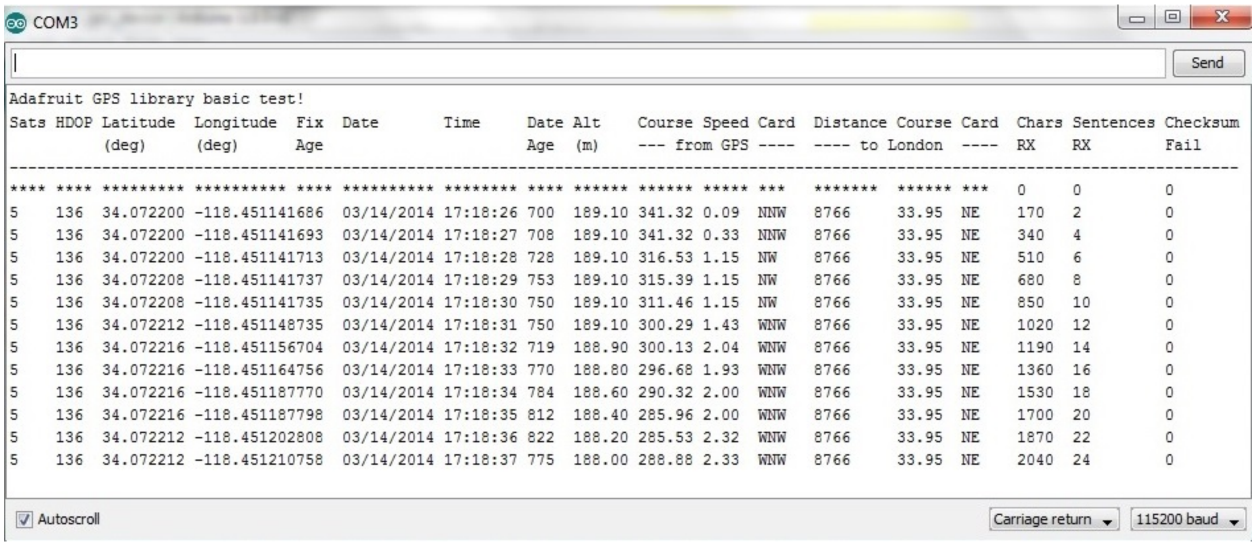
Hello Karen

Global Positioning System (GPS)

The GPS functionality is supported by an Adafruit Ultimate GPS Breakout – Version 3. The module has an internal patch antenna as well as a connector for an external active antenna, though we just use the former for this device. There exists an ENABLE pin on the module that, when pulled to ground, can turn off the module, as well as a red LED and pulse-per-second (PPS) output FIX that indicate if the GPS has made a fix on its location. When the module is still searching for satellites, the LED blinks at roughly 1Hz. Once it has established a fix, the LED blinks once every 15 seconds to conserver power [4].



Many of the SET commands one can use with this GPS module is encapsulated in functions provided by the Adafruit GPS library. The library also provides a parsing method to extract data, including the satellites used to find the fix on our location and the coordinates of that location. The figure below is a screenshot of some of that metadata formatted in a table for better comprehension.



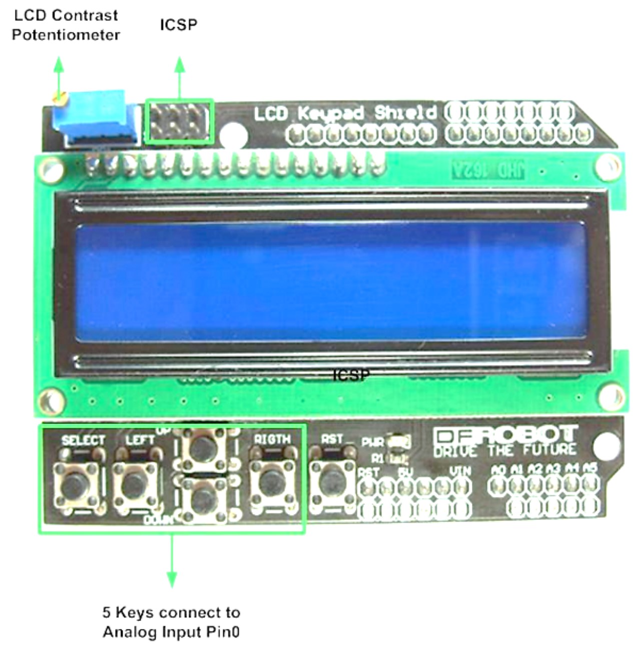
```
Adafruit GPS library basic test!
Sats HDOP Latitude  Longitude  Fix  Date       Time     Date Alt    Course Speed Card  Distance Course Card  Chars Sentences Checksum
          (deg)     (deg)      Age                      Age  (m)    --- from GPS ----  ---- to London  ----  RX    RX        Fail
-----------------------------------------------------------------------------------------------------------------------------------
**** **** ********* ********** **** ********** ******** **** ****** ****** ***** ***   *******  ****** ***   0     0         0
5    136  34.072200 -118.451141686  03/14/2014 17:18:26 700  189.10 341.32 0.09  NNW   8766     33.95  NE    170   2         0
5    136  34.072200 -118.451141693  03/14/2014 17:18:27 708  189.10 341.32 0.33  NNW   8766     33.95  NE    340   4         0
5    136  34.072200 -118.451141713  03/14/2014 17:18:28 728  189.10 316.53 1.15  NW    8766     33.95  NE    510   6         0
5    136  34.072208 -118.451141737  03/14/2014 17:18:29 753  189.10 315.39 1.15  NW    8766     33.95  NE    680   8         0
5    136  34.072208 -118.451141735  03/14/2014 17:18:30 750  189.10 311.46 1.15  NW    8766     33.95  NE    850   10        0
5    136  34.072212 -118.451148735  03/14/2014 17:18:31 750  189.10 300.29 1.43  WNW   8766     33.95  NE    1020  12        0
5    136  34.072216 -118.451156704  03/14/2014 17:18:32 719  188.90 300.13 2.04  WNW   8766     33.95  NE    1190  14        0
5    136  34.072216 -118.451164756  03/14/2014 17:18:33 770  188.80 296.68 1.93  WNW   8766     33.95  NE    1360  16        0
5    136  34.072216 -118.451187770  03/14/2014 17:18:34 784  188.60 290.32 2.00  WNW   8766     33.95  NE    1530  18        0
5    136  34.072216 -118.451187798  03/14/2014 17:18:35 812  188.40 285.96 2.00  WNW   8766     33.95  NE    1700  20        0
5    136  34.072212 -118.451202808  03/14/2014 17:18:36 822  188.20 285.53 2.32  WNW   8766     33.95  NE    1870  22        0
5    136  34.072212 -118.451210758  03/14/2014 17:18:37 775  188.00 288.88 2.33  WNW   8766     33.95  NE    2040  24        0
```

One issue with this module was that the GPS that is currently mounted on the shield is not responding to the power supply and hence does not turn on. I worked with Dr. Sarrafzadeh's lab to detach the module, but as of now, unsoldering the GPS module may damage the shield.

Consequently, a GPS module of the same model and manufacturer sits on a breadboard right now and is wired externally into the Serial3 pins on the Arduino.

Liquid Crystal Display (LCD) shield

The topmost shield of the PSD is a LCD screen with a keypad. This module has a blue backlight LCD displaying white characters. It consists of 6 keys – select, reset, up, right, down, and left – which are all connected to analog input pin A0. The pins on the upper edge of the shield include: a register select (RS) pin that controls where in the LCD's memory you write data to; a read/write pin (RW); an enable pin (E); display contrast pin (Vo); and eight data (D0-D7) pins that will be the bits written to the register. Additionally, there is a potentiometer that had to be adjusted with a screw driver to increase the contrast enough to display the segments on the screen clearly.



The Arduino IDE already comes preloaded with the LiquidCrystal library that supports the LCD keypad shield once the interface pins are assigned [5]. Using a digital multimeter, I identified which pins on the module corresponded to the parameters in the LiquidCrystal class constructor. Different shields have a different ordering of the pins, but for future reference, the configuration of our shield is to use `LiquidCrystal lcd(8,9,4,5,6,7).`

Memory Measurement

One of the main objectives of this quarter of the project was to be able to measure the available memory that would remain after loading all the necessary programs that support the modules analyzed in the previous sections. It is critical to know how much space is left on the device as it will help decide what needs modifying on the existing defense mechanism before it can be loaded onto the microcontroller [6]. The amount of flash memory used is easy to determine, as the IDE outputs that information when you compile a sketch. Meanwhile, EEPROM usage is based on the user's code, so it remains the same throughout the execution of the sketch.

There are more complex measurements made dynamically when analyzing SRAM usage. Consequently, I imported the MemoryFree library found in the Arduino Playground that had several methods of determining the amount of available SRAM [2]. The simplest version of the function is shown below.

```
int freeRam () {
  extern int __heap_start, *__brkval;
  int v;
  return (int) &v - (__brkval == 0 ? (int) &__heap_start : (int) __brkval);
}
```

Encryption Software

In order to do an initial test of a security feature on the Arduino, I imported the Advanced Encryption Standard (AES) library, a block cipher where the block size is 128 bits and key lengths of 128, 196, and 256 are used for several rounds of processing and encryption that include permutations, substitutions, and additions, among other operations. The experiments run were mostly meant to verify the results found in the work of a previous graduate student who ran similar memory metrics on AES and worked on the encryption side of the PSD [3]. The success of those tests indicates that it will be reasonable to load cryptographic software on this device and still have room to fit the other functionalities.

Future Work

As the project progresses, there are small modifications that will be made to some of the components I analyzed this quarter. For instance, we may want to change the Bluetooth pairing code from its default value. Eventually, it will also be necessary to replace the broken GPS module for this PSD to be complete and compact. With the addition of this GPS Breakout onto the Arduino, a potential addition to the security mechanism may take into consideration the location of the patient when their medical device receives commands. If we can obtain a fix on the location of the PSD and know the location of the home station, which is assumed to be stationary, GPS would be another means of detecting whether the medical device is receiving data from an authorized source, since it should only get such data when it is near the base station. The GPS breakout also can log data for up to 16 hours, which may prove useful for a tracking feature later on.

However, the largest changes in the project will most likely be in the code of the defense mechanism. Currently the code developed by Pournaghshband is on a full-fledged Linux machine with substantially more memory than an Arduino microcontroller, even if we were to mount a shield with additional RAM or use the storage on some of the modules attached to the shield. Next quarter, my plan is to go through the code, fully understand the algorithm, and see which areas can be optimized or substituted with more memory-conservative methods, so that the Arduino can accommodate both the defense mechanism and the existing modules on the PSD, moving us closer to the end goal of a fully functioning, portable PSD prototype.

References

1. "Arduino Mega 2560." *Arduino – ArduinoBoardMega2560*. 2014. <http://arduino.cc/en/Main/arduinoBoardMega2560#.UyNCiPldV8E>.
2. "Available Memory." *Arduino Playground – AvailableMemory*. 2014. <http://playground.arduino.cc/Code/AvailableMemory>.
3. Gupta, Priyansha. "Implementing Security in a Personal Security Device." 2013.
4. Ladyada. "Adafruit Ultimate GPS." 2013. <http://learn.adafruit.com/downloads/pdf/adafruit-ultimate-gps.pdf>.
5. "Liquid Crystal – 'Hello World!'." *Arduino – LiquidCrystal*. 2014. <http://arduino.cc/en/Tutorial/LiquidCrystal>.
6. Pournaghshband, Vahab, Majid Sarrafzadeh, and Peter Reiher. "Securing Legacy Mobile Medical Devices." *Wireless Mobile Communication and Healthcare, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, 61:163-172. 2013.
7. "Roving Networks Bluetooth^{TM} Product User Manual." 21 Nov 2011. <http://ww1.microchip.com/downloads/en/DeviceDoc/Bluetooth-RN-UM.pdf>.
8. "Roving Networks RN-171-XV 802.11 b/g Wireless LAN Module." 29 Oct 2012. <http://ww1.microchip.com/downloads/en/DeviceDoc/rn-171-xv-ds-v1.04r.pdf>.