PINPOINT: LOCATION BEACON AND TRACKING

BY EZEQUIEL LOPEZ III
UNDERGRADUATE OF COMPUTER ENGINEERING

ADVISED BY MR. JEFF GERFEN
PROFESSOR OF COMPUTER ENGINEERING

SPRING 2016

COMPUTER ENGINEERING SENIOR PROJECT

CALIFORNIA POLYTECHNIC STATE UNIVERSITY, SAN LUIS OBISPO

Contents

INTRODUCTION

PROJECT OVERVIEW

The purpose of Pinpoint was to create a device that can collect and transmit location information for multiple users on a wireless network. The device would be used to keep track of and communicate with other users nearby. The final design includes a touchscreen display as a graphical user interface (GUI), an XBee RF module for wireless networking, a GPS receiver for location tracking, and a Programmable System on a Chip (PSoC) to control the modules.

Although Pinpoint could have been implemented on other platforms – namely, as an Android or iPhone app – I purposefully chose to develop a completely self-contained and dedicated system to more accurately demonstrate my strengths as a student of Computer Engineering.

PROJECT GOALS AND OBJECTIVES

Goals:

- Allow a user to track the location of nearby users
- Allow a user to communicate with nearby users
- Provide an intuitive user interface for the system

Objectives:

- Use a GPS module to obtain user position
- Use an XBee module to communicate between users
- Use a touchscreen display to show information and receive user feedback
- Program a PSoC module to control the GPS, XBee, and touchscreen

PROJECT OUTCOMES AND DELIVERABLES

Outcomes:

- Foundation for future extensions of a location tracking system
- Demonstration of adequate knowledge in Computer Engineering
- Satisfaction of the senior project requirement for Computer Engineering

Deliverables:

- Multiple assembled system prototypes
- A video overview and demonstration of the system
- Program files for the software
- A design report to document the project progress

FORMAL PRODUCT DEFINITION

OVERVIEW

Pinpoint is an embedded system that can be separated into three distinct parts: location tracking, information broadcasting, and user interfacing.

Location is tracking is implemented using a GPS module to receive periodic updates of the user's location and storing that data for later use. Pinpoint then transmits and receives RF data between multiple other Pinpoint devices. To transmit information, the system uses an XBee module with a meshing network protocol. The data transmitted can be directed to certain users or broadcast to the whole network and may take the form of a textual message or binary data of the user's location.

The user interface allows the end-user to view the information received from the two modules and provide feedback. The user may manually input information (send messages) or toggle between different screens to view different types of data. To accomplish this, Pinpoint uses a touchscreen display with various information menus.

MARKETING REQUIREMENTS

1. The device shall be self-contained and portable
2. The device shall be simple to use
3. The device shall be able to obtain its location
4. The device shall be able to send and receive information to compatible devices
5. The device shall be controlled with a GUI

ENGINEERING REQUIREMENTS

1. The device shall determine its global position
   1.1. It shall determine its position at least once every three seconds
   1.2. It shall determine its position to within five meters of its true location while outside
   1.3. It should determine its position to within eight meters of its true location while inside
2. The device shall transmit its location to similar devices
   2.1. It shall transmit its location at least once every five seconds
   2.2. It should be able to transmit at least one mile while outside in an open environment
   2.3. It should be able to transmit at least one thousand yards while outside in an urban environment
   2.4. Devices shall be referenced by username
      2.4.1. Usernames might not be unique
      2.4.2. Usernames might not be user-defined, may be device assigned
3. The device shall display its own location and the location of surrounding devices
   3.1. It shall be able to display at least five surrounding devices
   3.2. It shall refresh device locations when received
   3.3. The display shall be user-interactive
      3.3.1. The user shall be able to view name and location information of nearby devices
      3.3.2. The user shall be able to send messages of at least 200 characters to specific nearby devices
      3.3.3. The user shall be able to broadcast messages of at least 200 characters to all nearby devices
4. The device shall log user data
   4.1. Log entries shall include the username, position, and timestamp
   4.2. It shall log at least ten minutes worth of data
   4.3. It shall log at least two minutes worth of nearby devices' data
5. The device shall be portable
   5.1. It shall run on a battery pack
      5.1.1. The battery pack shall last at least five hours during use
      5.1.2. The battery pack shall be rechargeable
   5.2. It shall not weigh more than two pounds
   5.3. It shall not measure more than thirty square inches

CRITERIA

The following criteria, for which the engineering requirements will direct the development of Pinpoint, have been listed in order of highest priority to lowest.

Interface Usability:

Usability can be separated into success rate, time per task, error rate, and users' subjective satisfaction. High success rate, low time per task, low error rate, and high satisfaction improve the user experience.

Communication Distance:

Large transmission distances are needed to ensure consistent contact while users are farther from each other.

Battery Run Time:

Extended/Long battery life is ideal to allow for prolonged communication.

Location Accuracy:

Greater location accuracy is desired for more precise user navigation.

DESIGN AND JUSTIFICATION

PROCESS OVERVIEW

The overall design for Pinpoint began as a list of top-level functionality that the device as a whole would be able to accomplish. Next to each objective listed, the hardware modules required to complete each objective were listed. For example, the first two function objectives listed were obtaining and sending out GPS location for the user. The first of these required a GPS module and the second needed both a GPS and RF module.

Once the two lists were completed, a compacted version of the hardware list was created and used to determine what modules would be necessary. From this list the four basic hardware modules were chosen: RF, display, GPS, and microcontroller (MCU). At this point I performed extensive research for each module to determine which products provided the best functionality while keeping in mind the level of implementation difficulty. Once the research was finished, I purchased the products that would fulfill each hardware module.

After the list of hardware modules was finalized, I began designing the overall system. For the hardware portion, I drew block diagrams to determine what the hardware interfaces would need to look like. Knowing how the hardware connected together, I was then able to begin designing the software that would control everything.

The software architecture was created using the top-down design method and a software dependency tree began taking shape. This tree was generated by reversing the method used to determine the hardware modules. I took the hardware list and mapped them to the top-level functionality they would need to implement. I categorized the objectives and expanded on what duties they would need to perform in order to complete their goals.

From then on I proceeded with bottom-up implementation, meaning I took the list of categorized objectives and began implementing them one-by-one. For each objective function I performed enormous amounts of testing and checked against the software dependency tree to build it from the leaves to the root.

After finalizing the hardware, and near the end of implementing the software, I began designing a prototype for the mechanical design. With consideration to the requirements for maintaining a self-sufficient device, I designed a schematic that included all the hardware modules and necessary peripheral circuits. From this schematic I ordered the required parts and began designing the printed circuit board (PCB) for the device. With that, Pinpoint's design process was complete.

The project can be modeled in two overall diagrams: on the network level and on the device level. The network level, shown in Figure 1, consists of the RF connections each device has to each other. All Pinpoint devices can communicate with each other through the XBee modules, which run on the IEEE 802.14.5 Zigbee Protocol. This means that, within a Pinpoint network, there must be exactly one coordinator (User 1) associated with any number of endpoints (Users 2 and 3). Consequently, the ease of system design, in terms of network connection, comes at the expense of having to curate the network beforehand. Each device also supports RF communication with GPS satellites. As shown, the devices are only capable of receiving information from the satellite and not transmitting anything back.

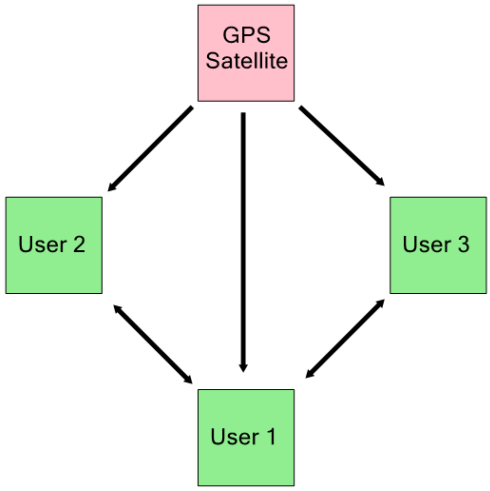

Figure 1: An example of a Pinpoint network-level diagram.

The device level, shown in Figure 2, is made up of four major modules: a PSoC, a GPS receiver, an XBee device, and a touchscreen display. As explained earlier, the GPS and XBee connect to the network level and communicate with other Pinpoint devices. However, the display is for direct user input and output. The LCD displays the information needed and the user touches control which information gets shown. Each of the three modules communicate with the PSoC so that it may manage all the information it receives and act on it accordingly. More information on how the PSoC manages the modules can be found in the Software Architecture section.

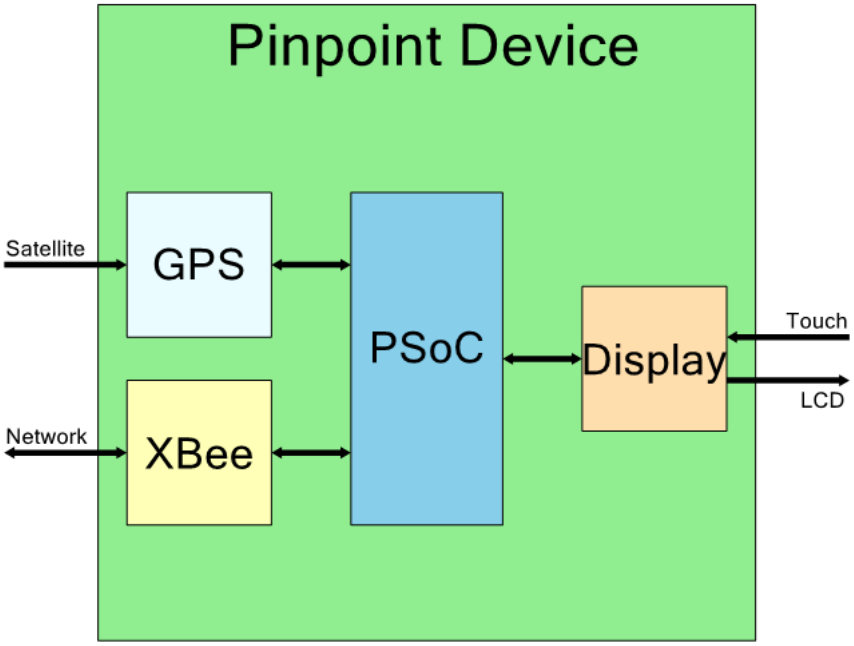

Figure 2: The device-level system diagram.

The hardware diagram is fairly simple for a Pinpoint device and can be seen below in Figure 3. Since the system must carry its own power supply and certain modules require specific voltages, the device needs a voltage regulator. This regulates the varying battery voltage (from fully charged to discharged) to the 3.3V needed for the XBee and 5.0V needed for the other modules.

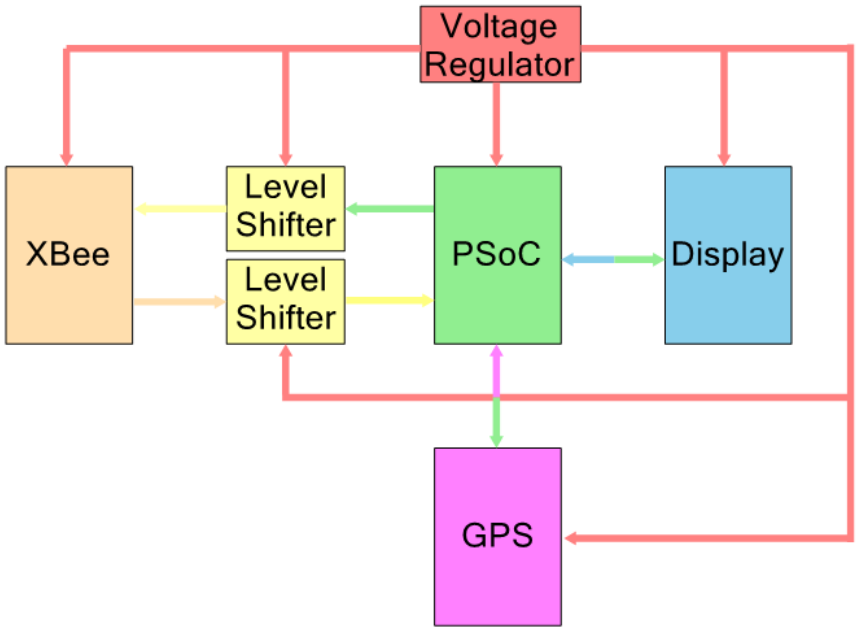

Figure 3: The hardware block diagram for the Pinpoint device.

Since the XBee runs on different power and logic levels from the PSoC, level shifters are needed in between to convert the 5V level from the PSoC to the 3.3V to the XBee, and vice versa. The reason there are two level shifters is because the XBee runs on the RS-232 communication standard and requires both RX and TX lines. Thankfully the other two modules are much simpler since they both run on the same logic levels.

The GPS module also uses RS-232 for communication with the PSoC, but for simplicity it is shown as having a single bidirectional data line. The display module is actually fairly simplified in the schematic. The LCD screen itself is run by a RA8875 graphics chip (for ease of implementation) which uses the SPI communication standard. The RA8875 chip requires MISO, MOSI, and SCK lines, plus an additional few for touch information.

HARDWARE COMPONENTS

Voltage Regulator:

The voltage regulator module can be broken down into two circuits: one outputs 5.0V and the other 3.3V. Each circuit uses a LM317 chip, a 240Ω resistor, and a 1kΩ trimmer. To get a

3.3V output, the trimmer is set to approximately 390Ω, and set to about 680Ω for 5.0V output. Each regulator is capable of supplying 1.5A to the load.

Level Shifter:

The level shifters also utilize three components to obtain their output. Two 10kΩ resistors bias an N-type enhanced mode MOSFET against both logic levels voltages so that input to the drain at 5V logic exits the source at 3V logic and vice versa. Each shifter circuit consumes at most 1 mA.

XBee:

The XBee is a special case when it comes to power supply. It runs on and communicates with 3.3V logic using the RS-232 communication standard. The difference in logic levels requires the use of logic level shifters, as described above, between XBee and PSoC.

XBee implements the RF communication needed to create the device network. The XBee consumes up to 220 mA, 62mA, and 15 mA during transmit, receive, and idle operating states, respectively. With estimated operating times of 50 mS, 300 mS, and 650 mS for each respective operating state per second (depending on the number of users connected), the XBee will have an average current consumption of about 40 mA with peaks up to 220 mA.

PSoC:

The PSoC uses 5V for power and logic levels. There are scant details about power consumption in the device documentation, but experimentally, the device has an average current draw of 28 mA.

Display:

For simplicity, the display has been shown as a single unit in the figure above. In actuality, the LCD touchscreen is controlled by a RA8875 driver board. The driver board handles voltage regulation and logic level shifting for the screen's controller. The screen uses approximately 0.33 mA while the driver board consumes from 30 mA to 155 mA depending on the brightness settings for the screen. The default setting uses minimum current.

GPS:

The GPS module runs on 3.3V power and logic-level, but comes with a built-in voltage regulator and level-shifters. It consumes 25 mA during position acquisition (device startup) and only 20 mA during tracking. For outdoor use, acquisition will only be needed for about 35 seconds when the GPS module first powers on, therefore power usage can be estimated at 20 mA. Indoor or urban usage, though, will extend the acquisition time depending on overhead cover. Power consumption in this case can be estimated at 22 mA, assuming the GPS module spends half of its time acquiring satellite signal.

The total current draw for Pinpoint can be estimated to be the sum of its component parts:
2 x 1 mA + 40 mA + 28 mA + 30 mA + 22 mA = 122 mA

HARDWARE DESIGN DECISIONS

The important hardware choices for Pinpoint included finding the right modules that can handle the demand. To start, there had to be a controller unit that could handle I/O from different modules using a variety of communication standards at close to real-time. It would also need to be able to store and manipulate different data structures to keep track of users.

The first controller that came to mind was the Arduino – which I actually began development for. Quickly, I found that the simplistic language on which it ran could not easily handle the data structures needed for the project. I then thought of using the Raspberry Pi, but dismissed it because of the overhead required to get the project running. After using the PSoC in a different project, I chose to use it for Pinpoint because of the simple hardware/software interfacing and immensely useful integrated development environment (IDE). The PSoC also had more than enough interface pins and a relatively low hardware profile, which made it the clear champion.

After deciding on the controller, the next big module to choose would be for RF communication. Pinpoint required a module that could communicate to multiple end-users and had a reasonably large transmit radius. The XBee module immediately came to mind because it proved itself to be simple to set up and use in previous projects. Although that still holds true, there is one downside to using XBees for this project: the network must be curated such that there is exactly one "coordinator". Apart from this drawback, the XBee provided everything needed for the RF network.

The final, and simplest, module to choose was the GPS. Nearly all GPS chips can provide the required accuracy for this project and use the same National Marine Electronics Association (NMEA) string output format. The main factor in deciding on a certain chip was the ease of use. The Adafruit module rose above the rest because of its built-in features and the use of RS232 for communication.

EVALUATION OF HARDWARE DESIGN CONCEPTS

As a one-off project, the hardware design stands as a good template for further versions. As a marketable product, though, there are a few considerations that need to be taken into account for each module. With the PSoC, the physical size of the module is a major drawback. Apart from the touchscreen, the PSoC used the highest amount of area on the PCB. This is because the PSoC kit is designed to be breadboard-friendly with 0.1" spacing between pins. To reduce the footprint for the PSoC, instead of using the prototyping kit, the 68-pin QFN chip could be used directly. This will, however, increase the complexity of the hardware design to account for the loss of amenities (e.g. power regulators, crystal inputs, programmer connections, etc.).

The display driver, GPS, and XBee modules follow similar limitations. Since they connect with through-hole pins, the modules stand a fair distance away from the PCB plane. To

mend this issue, the individual components could be purchased and placed directly on the main board.

The XBee, though, has a unique limitation of its own: it has a wire antenna that sticks out for RF reception. The module, then, would have to be replaced with a version that has a ceramic antenna, similar to the GPS, to reduce the vertical footprint.

The software for Pinpoint used top-down design and bottom-up implementation which allowed frequent, incremental testing of each unit. A visual representation of the dependency of each software unit on the others is shown in Figure 4 below. Following a logical, intuitive design model, the top four software components of the dependency tree represent the four corresponding hardware components.

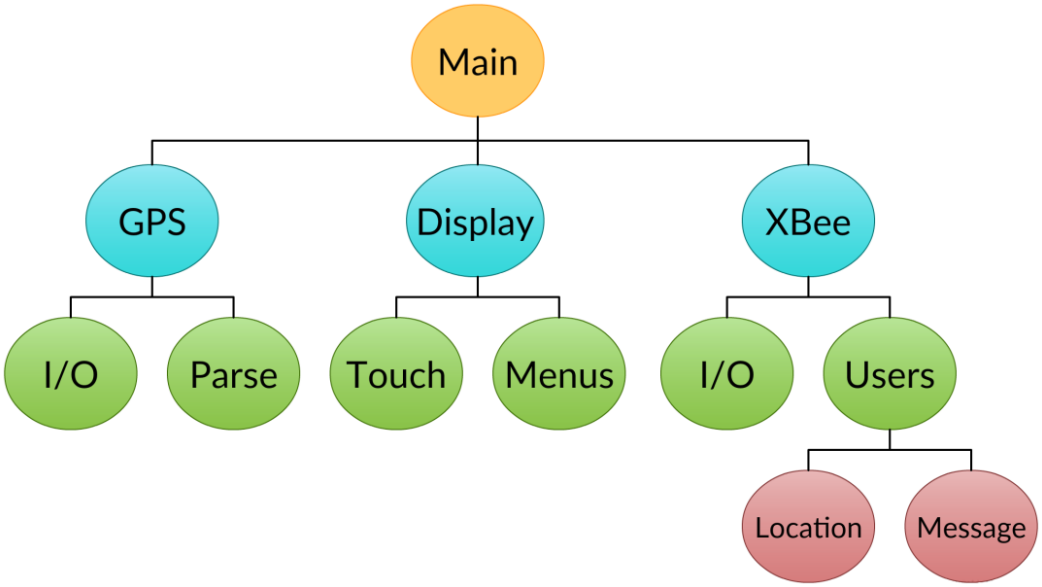

Figure 4: Software dependency tree for Pinpoint.

Top-down design establishes that each component does less "work" than the components below and that the "work" becomes more specific at lower levels. The software for Pinpoint follows this technique by having each parent in the dependency tree delegate actions to a matching child component. For example, if a transmission were to come in from another device, Main would have XBee perform the hardware I/O to retrieve and decode the transmission and update the User records to reflect the new location information or message received.

Bottom-up implementation was the style chosen to write the software. It pairs perfectly well with top-down design because there is a clear understanding of what each leaf will do and why it is necessary. If proper testing is done on each child of the dependency tree, debugging during implementation can be focused on the parents that utilize them. To illustrate bottom-up implementation, the very first function I wrote for Pinpoint was a parser for NMEA strings that stored the information in simple data structures. After extended testing and verification, I used that function to prove that I/O from the device was being received properly by checking for valid data in the structures.

Main:

Main, being the heart of Pinpoint, is responsible for maintaining order and assigning work to each of the components below it. Main is the owner of the core data structures – including the User list and location information – but does no work to maintain them. The only real "work" Main takes part in is initializing the components so they can update the structures and keep track of different states. After initialization, Main endlessly loops, waiting for any input that requires action and allots it to the proper component.

GPS:

The GPS component is a bit more interesting than Main in terms of responsibilities. GPS is appointed to be the interface between the PSoC and GPS modules. One of its key duties includes managing I/O communication. To do this, Pinpoint utilizes a UART component in the PSoC Creator IDE to handle RS232 transmissions. For sent data – only used in initialization – GPS uses the UART component to output the proper settings for the GPS chip. For received data, GPS buffers the NMEA sentences and raises a flag when there is a complete sentence that is pending. When the ready flag is raised, GPS is instructed to parse the NMEA string depending on the sentence type. It then returns the structured data, which Main stores for later use. More information and documentation for NMEA sentences can be found at freeNMEA.net.

Display:

Display is the most "visible" component of the project because it represents the user interface. Display is tasked with ensuring that interaction with the touchscreen is natural and behaves as expected. Its first duty is to detect user touches and notify Main when there is a new touch which needs a response. Display provides the coordinates of the touch response which are then given to the Menu component. Using the touch coordinates, Menu determines if the touch warrants a response and, if so, displays the correct menu on the screen. Each menu has a set of buttons which the user can touch to navigate between screens. More information on the operation of the Menu system can be found in the Behavior Models section.

XBee:

Being the only component to have leaves in the fourth level of the dependency tree, XBee proved to be one of the most complex components to implement. XBee manages I/O communication the same as GPS, using a similar UART component, and raises a flag to indicate a pending transmission. When Main encounters the flag, though, it does not know – or care – about the contents of the transmission. During the parsing process, XBee manages the User list making sure to create new structures for new users and updating those structures with received data. The data in each received transmission is parsed and stored in the proper user's structure, independent of oversight from Main.

Each device also periodically broadcasts an update of its own information onto the network to every other device. This is achieved using a timer component in the IDE to raise a flag when it is time to transmit. Main detects the raised flag and instructs XBee to send the update.

Users:

Although it is not one of the top four software components, Users is especially important because it contains and manipulates all of the data for each friend seen on the XBee network. Whenever a new friend is detected, a new structure is created for them and is populated with the standard information contained in a location broadcast: username, unique identifier, and position. The username can be up to twenty characters long and is specified by the user. The unique identifier, however, is specified by the PSoC module and was written to the device during manufacturing.

While interacting with a friend, the User structure gets populated with other types of information, apart from location. Each friend's structure contains a list of messages that have been sent between the friend and the user. This Message structure contains the message text and header information such as the length, the originator, and links to the next and previous Messages in the list. Each User structure also has a message buffer so that the user may pause and resume an unfinished message at any time.

BEHAVIOR MODELS

GPS:

The top-level purpose for the GPS component is to take I/O from the GPS chip and store it in an easily accessible structure. To do so, the software traverses a flowchart to determine what its next course of action should be. A visual representation of the flowchart is shown in Figure 5 below. To begin, GPS waits either for data to be received at the hardware interface or for the sentence-ready flag to be raised.

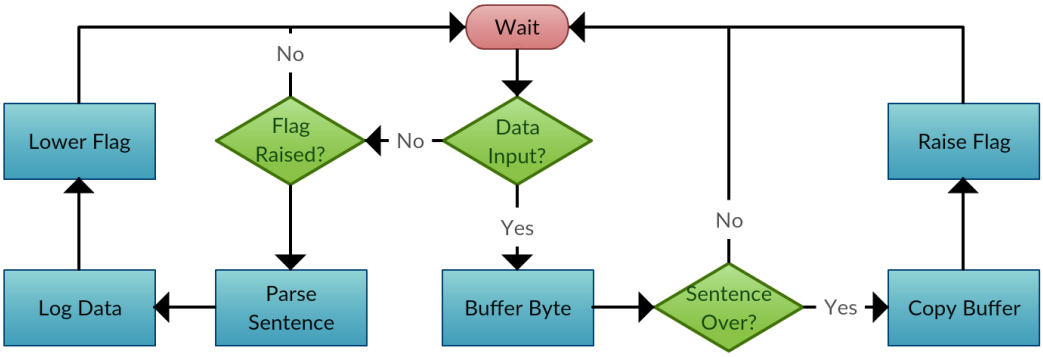

Figure 5: Behavior model for the GPS component.

If there is data from the chip, GPS stores the byte at the end of a temporary buffer. If the stored byte does not complete a NMEA sentence, GPS returns to the Wait state until more

data arrives. If the sentence is over, however, the data gets copied into a secondary buffer and the sentence-ready flag is raised. Because sentence buffering occurs within an interrupt service routine (ISR), GPS uses a double-buffer system to be able to assemble incomplete sentences while simultaneously parsing complete sentences.

Once the sentence-ready flag has been raised, GPS parses the message depending on which type of sentence it is. This GPS component currently supports GGA, GSA, GSV, RMC, and VTG sentence types with unique data structures for each. After parsing the sentence, GPS stores the data into a structure for Main and lowers the sentence-ready flag.

XBee:

XBee follows a very similar pattern to GPS because its purpose is nearly the same. XBee is tasked with controlling communication with the RF network through the XBee module and storing all received data. The behavior model for XBee, seen below in Figure 6, shows a data buffering system similar to GPS. All information is stored in a preliminary buffer until the transfer is complete, at which point, the data is copied into a secondary buffer and the data-ready flag is raised. For XBee transmissions, each message has a header that includes the length of the payload, so checking for the end of the message only requires knowing the amount of bytes received.

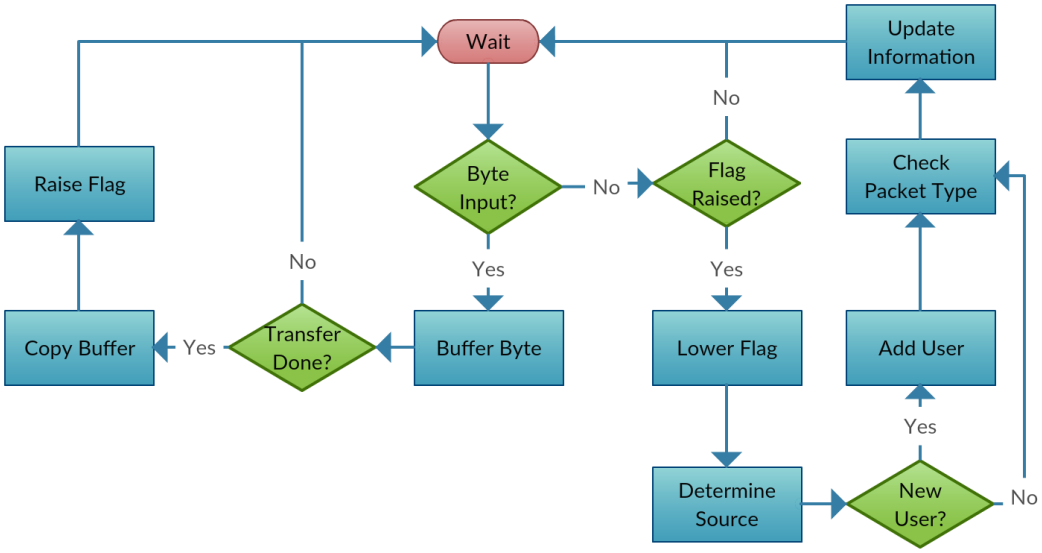

Figure 6: Behavior model for the XBee component.

The differences between XBee and GPS are very clear when it comes to parsing the data. When XBee detects that the data-ready flag is raised, it lowers it and checks the unique ID of the message originator. If XBee does not find a User in the friends list with a matching ID, it creates a new User to populate the data into. If a matching User is found, it skips the User creation and goes directly to checking the packet type. XBee can determine the type of data in the payload by reading the header information and then it stores the data in the proper structure. For a location broadcast, XBee overwrites the old information in a User's structure with the new location data. If the receive transmission was a message, XBee adds

the message information into a new Message structure and places it at the end of the friend's message list.

Display:

Although Display is made up of both Touch and Menu components, Touch is somewhat trivial so most of the behavior model for Display will revolve around Menu. Touch is considered trivial because it is a stateless component that requires hardly any extra software to complete its purpose. Touch's only objective is to determine whether the user has touched the screen and report the coordinates. Thankfully, the display controller module keeps track of touch data and Adafruit supplied a function that reads the touch information from the module. Touch, then, is implemented by polling the display controller for touch data and reporting it back to Display. The only reason why Touch is mentioned, then, is because, without this component, there would be no way for Menu to receive its required input. It is from a confirmation of touch activity, then, that Menu is run.

For the visual representation of Menu's behavioral model, shown in Figure 7 below, I have taken the liberty of simplifying the diagram. First, all red arrows indicate the user touched the "Back" button on the screen. Second, the black arrows indicate the user touched the corresponding button on the screen that matches the destination. Finally, black arrows can be optionally accompanied by a condition statement surrounded by square brackets.

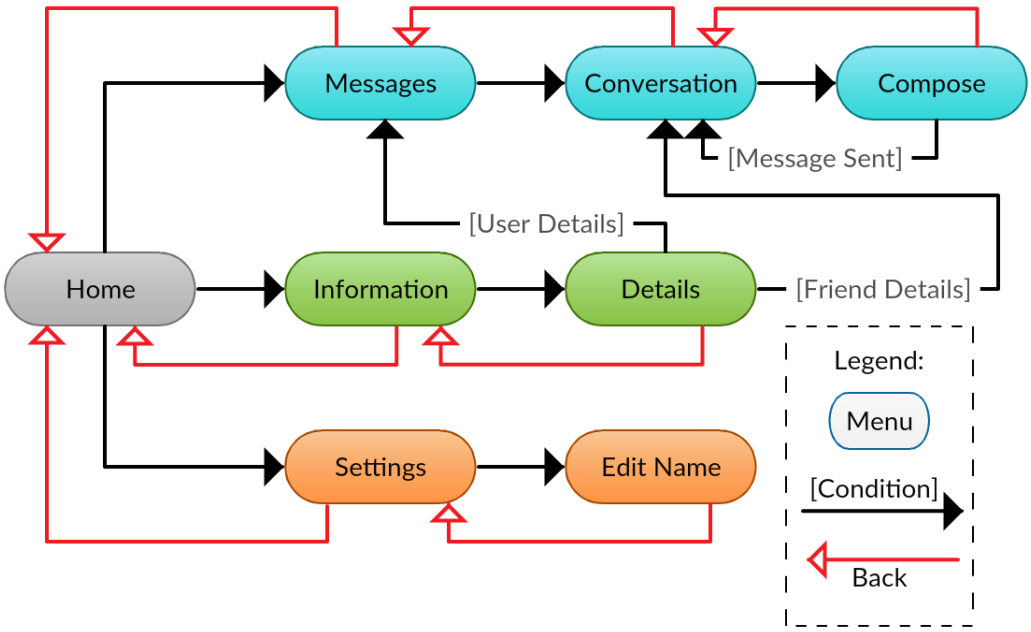

Figure 7: Behavior model for the Menu component.

At device startup, Home is displayed as the default menu. This screen shows the username and time at the bottom along with a map of nearby users labeled as dots. At the top there are three buttons that lead to the Settings, Messages, and Information menus. All menus apart from Home have a "Back" button that leads to its parent menu.

The Settings menu instinctively shows all of the settings that the user is able to control. The first is the username which, if touched, will lead to a sub-menu that allows the user to type their desired name with an on-screen QWERTY keyboard. In this sub-menu, when the user is finished editing their name, they may hit the "Done" button. The other two user-definable settings are the GPS update rate and the XBee transmission rate. The GPS update rate changes how often the GPS module reports the user's position, while the XBee transmit rate changes the frequency of location updates sent out onto the XBee network. The user can tap the update rate they prefer for each setting and the Pinpoint device will adjust accordingly.

The second menu accessible through the Home screen is Information. This menu lists the usernames for the user and the other "Friends" that have been seen on the XBee network. If the user touches any of the names listed the Details sub-menu will be displayed, showing the specific information for the selected user. Underneath the user details is a "Messages" button that will lead to one of two menus depending on the user selected. If the details shown are for a friend that has been seen on the network, the "Messages" button leads to the Conversation menu for that specific friend. If, however, the details being shown are for the end-user, the "Messages" button leads to the Messages menu.

Messages is the final menu that is accessible through the Home screen. Similar to Information, Messages shows a list of usernames on the screen, but only those of friends, not the end-user. Next to each username is the number of messages sent between the user and the friend, surrounded by parentheses. If the user touches one of the listed usernames, the Conversation menu is shown, which lists the most-recent messages. Because of screen restrictions, only the maximum number of messages in reverse-chronological order that fit on the screen are shown. Below the list of messages is a "Compose" button that leads to the Compose menu that allows the user to type a message to the friend. Similar to Edit Name, Compose has an on-screen QWERTY keyboard for the user to compose messages. However, each message is limited to 240 characters. Once a user is finished composing their message, they may tap the "Send" button, and the message will be sent over the XBee network to the friend.

SOFTWARE DESIGN DECISIONS

The most important decision for the software architecture of Pinpoint was to use the top-down design method. It was a key instrument for keeping the architectural layout as simple as possible without sacrificing functionality. To more concretely explain the value of top-down design, we should envision Pinpoint's alternate structure without proper architectural hierarchy.

One common method of implementing programs is to list all of the things the project should be able to accomplish and write function prototypes for them. This would leave a heap of functions that need to be implemented and no real idea where to begin. In the context of Pinpoint, it would be a flattened version of the software dependency tree: twelve components that all require simultaneous implementation. If the overwhelming sense of

disorganization was disregarded and the software writing began anyway, there would be frequent dead-halts during development because of improperly or incompletely tested functions. This is what top-down design paired with bottom-up implementation aims at avoiding.

EVALUATION OF SOFTWARE DESIGN CONCEPTS

With every file and function written for Pinpoint, the main goal was to write code that I, myself, would not mind maintaining. In this goal, I was fairly successful. I believe any other (fairly proficient) programmer would be able to create a very similar design diagram if they were handed the source code. All functions that pertained to certain components in the dependency tree were separated into corresponding source files to further emphasize the idea that they operated in unison.

I do concede that certain components, namely Menu, could have been written in a more compact, procedural manner. Currently, Menu relies on fifteen functions and 750 lines of code to display the proper information on the screen, compared to an average of about four functions and 200 lines of code for the other components. Code inflation of this level could be attributed to the fact that Menu reaches into the data from the other components to fulfill its purpose. Although this does not justify such distension, it does imply the need for more programs to fulfill the increased complexity.

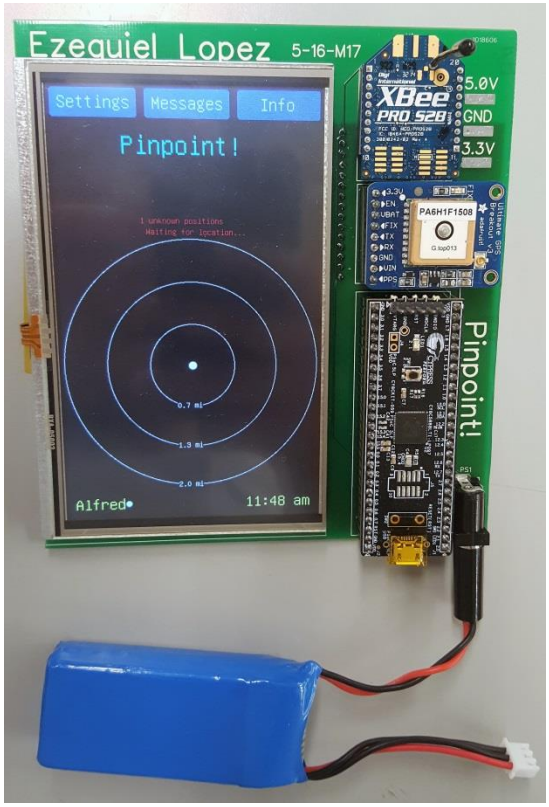

Figure 8: Final mechanical design for Pinpoint.

Since the list of major hardware modules has already been finalized by this point, the only extra step needed to complete the electrical schematic for Pinpoint was to determine the peripheral hardware that was critical to meeting the engineering requirements. The main constraint to be met was to make Pinpoint portable.

To do this, Pinpoint would need a battery of at least 610 mAH to meet the five hour minimum run time with a calculated current draw of approximately 122 mA. Apart from that one detail, everything else had been planned out with the design of the hardware architecture. For more information on the hardware components and connections, the device schematic can be found in the Additional Diagrams section of the Appendix.

The next step was to order the remaining parts and design a PCB to connect them all together. The component footprints and PCB were designed using a program called DipTrace and the PCB design was sent for manufacturing. The final PCB designs, along with the Bill of Materials, can also be found in the Additional Diagrams section of the Appendix.

DESIGN DECISIONS

When ordering parts for the final mechanical design, the main decision was whether to use through-hole or surface-mounted (SMD) components. Because one of the requirements states that the device shall measure less than thirty square inches, and the device has to be

mechanically stable due to being portable, I was leaning toward using SMD components. The difference in board space used for SMD and through-hole components turned out to be a non-issue for this project since those components would sit underneath the larger modules. Mechanical stability was also a moot point since this version is only a prototype and won't be mistreated. I, therefore, designed both versions to test the differences in manufacturability and design nuances.

One seemingly strange decision I made for the mechanical design was to make each of the major hardware modules detachable. It is an easily overlooked situation, but at some point during testing or use a module can (and most likely will) stop working, for whatever reason, and need to be replaced. Learning from previous experience, replacing dead hardware that has been soldered in twenty places is no easy task. For this reason I designed the modules to be easily detachable so that malfunctioning hardware can be replaced without impeding further testing and implementation.

The final design decision for the PCB was to add voltage breakout tabs. These tabs provide direct connection to 5V, 3.3V, and 0V nets purely for ease of voltage adjustment. To adjust the output of the voltage regulators without these tabs, one would need to find small test probes and hold them in the plated through-holes while adjusting the trimmers. With these tabs, one could simply place alligator clips on top and adjust the trimmer without worrying about short-circuiting anything.

SYSTEM INTEGRATION AND TESTING

TEST PLAN

The following tests are designed to check Pinpoint's adherence to the engineering requirements that have guided this project's implementation. They will test each requirement alone, or combined with others, to achieve a comprehensive sense of project completeness.

Test	Requirements	Procedure	Pass Condition
1	1.1	Run the device from startup for one minute and count the position fixes.	20 or more fixes
2	1.1	Count the position fixes for one minute after the device has been running for one minute.	20 or more fixes
3	1.2	While outside, record the device position after running for one minute.	Recorded location at most 5 meters from true location
4	1.2	While outside, record the device position after running for five minutes.	Recorded location at most 5 meters from true location
5	1.2	While outside, record the device position after running for ten minutes.	Recorded location at most 5 meters from true location
6	1.3	While inside, record the device position after running for one minute.	Recorded location at most 8 meters from true location
7	1.3	While inside, record the device position after running for five minutes.	Recorded location at most 8 meters from true location
8	1.3	While inside, record the device position after running for ten minutes.	Recorded location at most 8 meters from true location
9	2.1	Run the device from startup for one minute and count the location transmissions.	12 or more transmissions

Test	Requirements	Procedure	Pass Condition
10	2.1	Count the location transmissions for one minute after the device has been running for one minute.	12 or more transmissions
11	2.2	In an area free from buildings and geological barriers, separate two devices by one-quarter mile, check network connectivity, and repeat until communication fails.	Maximum transmission distance at least one mile
12	2.3	In an area with buildings and/or geological barriers, separate two devices by 200 yards, check network connectivity and repeat until communication fails.	Maximum transmission distance at least 1,000 yards
13	2.4, 3.1	Run five devices in close proximity and check the map display after five minutes.	Each display shows 5 nearby devices
14	2.4, 3.3.1, 4.1	Run at least two devices in close proximity and check the details for each device after five minutes.	Each device displays the correct details for each nearby device
15	3.3.2	Run two devices in close proximity and send a message of 200 characters from each device to the other.	Both devices successfully send, receive, and display the messages
16	3.3.3	Run at least three devices in close proximity and send a broadcast of 200 characters from each device.	All devices successfully send, receive, and display the broadcasts
17	4.2	Run a device for ten minutes and check the location data.	There is location data for the past ten minutes
18	4.3	Run at least two devices for two minutes and check the location data for each nearby device.	There is location data for the past ten minutes for each nearby device
19	5.1.1	Run a device on a full battery until it powers off.	The device powers off after five hours

Test	Requirements	Procedure	Pass Condition
20	5.2	Measure the weight of the device.	The device weighs less than two pounds
21	5.3	Measure the area of the PCB.	The area measures less than thirty square inches

RESULTS/ANALYSIS OF VERIFICATION

	Result	Pass/Fail	Notes
1	30 fixes	Pass	
2	30 fixes	Pass	
3	2.16 m	Pass	Measured: 35.2999, -120.6607 Actual: 35.299909, -120.660719
4	2.16 m	Pass	Measured: 35.2999, -120.6607 Actual: 35.299909, -120.660719
5	2.16 m	Pass	Measured: 35.2999, -120.6607 Actual: 35.299909, -120.660719
6	N/A	**Fail**	Position not acquired
7	4.88 m	Pass	Measured: 35.3000, -120.6610 Actual: 35.300038, -120.661028
8	4.88 m	Pass	Measured : 35.3000, -120.6610 Actual: 35.300038, -120.661028
9	15 transmissions	Pass	
10	15 transmissions	Pass	
11	>1.75 miles	Pass	From: 35.291901, -120.689160 To: 35.273400, -120.710321
12	>1360 yards	Pass	From: 35.302405, -120.657136 To: 35.305475, -120.670323
13	N/A	Pass	Because of lack of funding, 4 virtual devices were added to the network.
14	N/A	Pass	
15	N/A	Pass	

	Result	Pass/Fail	Notes
16	N/A	**Fail**	Broadcasting not implemented
17	N/A	**Fail**	Long-term logging not implemented
18	N/A	**Fail**	Long-term logging not implemented
19	3 hrs. 45 min.	**Fail**	
20	1.8 lb.	Pass	
21	25.4 $in.^2$	Pass	

Implementation against the design requirements turned out nearly perfect. Most of the tests were passing, and from the ones that failed, three tests failed because they weren't implemented, but the other two failed because of legitimate design flaws. For instance, the battery test failed because the calculated power usage was a lot lower than what it actually turned out to be. To pass the battery requirement, though, the fix is as simple as purchasing a battery with a higher capacity.

Although not all of the tests were passing, I would consider the project a success. The core functionality of Pinpoint worked well, and some aspects, specifically the transmit distance, exceeded expectation.

CONCLUSION

Pinpoint has been the most involved project that I have worked on to date. It required a large portion of my time in the six months I was given to complete it, but I have found it to be entirely satisfying. One of my main goals when starting Pinpoint was to exhibit the amount of knowledge and experience I have accumulated throughout my undergraduate program in Computer Engineering. I believe I have been completely successful in this respect because of the many areas of knowledge that I needed to delve into in order to reach the conclusion.

Beginning this project was a very daunting task because I had so many ideas and expectations about the final result that it was hard to find somewhere to start. Thankfully, my advisor, Professor Gerfen, set me on the right path by having me draw out the various black-box diagrams and write a list of requirements. This was such an instrumental foundation for the rest of the project and it lead me to be methodical and comprehensive during design and implementation.

As for the final outcome of Pinpoint, I am proud of the device I have created because it very closely resembles what I had envisioned at the start. There are, admittedly, a few portions that I wish I had more time to complete and perfect. The messaging system, as mentioned in the requirements and tests, was supposed to allow broadcasts through the network, but there was no time to add that feature. The initial set of project goals had also included a lengthy list of optional functionality – like waypoints and geo-fencing – that I knew I would not be able to include, but the final version of Pinpoint captured the essence of the idea.

For a more tangible perspective of Pinpoint and to experience how the hardware behaves in actuality, a video overview and demonstration can be seen here. To explore the contents of the project code or to build Pinpoint yourself, the source files are hosted in this repository.

REFERENCES

DePriest, Dale. "NMEA Data." GPS Information. Joe Mehaffey and Jack Yeazel, n.d. Web. 15 Jan. 2016.

Digi. *XBee/XBee-PRO ZigBee RF Modules User Guide*. Dec. 2015. Rev. X.

GlobalTop Technology Inc. *FGPMMOPA6H GPS Standalone Module Data Sheet*. 31 Jan. 2012.

IEEE standard for local and metropolitan area networks. New York: Institute of Electrical and Electronics Engineers, 2011. Print.

Nielsen, Jakob. "Usability Metrics." Usability Metrics. Nielsen Norman Group, 1 Jan. 2001. Web. 21 Jan. 2016.

RAiO Technology Inc. *Character/Graphic TFT LCD Controller Specification*. 16 Oct. 2014. Version 1.9.

"Top-Down Design." University of Maryland, Baltimore County, n.d. Web. 15 June 2016. <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=21&ved=0ahUKEwj69tvRwbrNAhWDKGMKHUo6C7AQFgiCATAU&url=http%3A%2F%2Fwww.csee.umbc.edu%2F~stephens%2F104%2FPPT%2FL22Top-DownDesign.ppt&usg=AFQjCNFfbnUuoIc2OBn8OCJ2ofc1tZVIHQ&sig2=CKiWztfHYz2WgUYb009cQQ&cad=rja>.

BILL OF MATERIALS

Table 1: Bill of materials for the through-hole version of Pinpoint

Part #	Name	Description	Qty	Unit Price	Cost
CY8CKIT-059	PSoC 5LP	Development Boards & Kits - ARMCY8CKIT-059 PSoC 5LP Dev Kit	1	$9.97	$9.97
1590	Touch screen driver	RA8875 Driver Board for 40-pin TFT Touch Displays	1	$34.95	$34.95
1596	Touch screen display	5.0" 40-pin TFT Display - 800x480 with Touchscreen	1	$39.95	$39.95
746	GPS receiver	Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3	1	$39.95	$39.95
803063	Rechargeable Li-Po Battery	7.4V 1200mAh	1	$10.88	$10.88
N/A	Printed circuit board	FR-4 .062 1 Oz Cu, Up to 50 Square Inches	1	$30.00	$30.00
171-PA5525-1-E	DC Power Plug	2.5 mm power connector	1	$1.14	$1.14
161-0725-E	DC Power Socket	2.5 mm power socket	1	$1.14	$1.14
929834-02-36-RK	36 pin header stick	2.54 mm pitch header stick	2	$1.84	$3.68
M20-7820942	9 pin socket	9 PIN SIL VERTICAL SOCKET GOLD	1	$1.20	$1.20
M20-7821546	15 pin socket	15 PIN SIL VERTICAL SOCKET TIN	1	$1.51	$1.51
929974-01-26-RK	26 pin socket	26 CON STR BRDMNT SKT	2	$2.25	$4.50
M22-7131042	10 pin socket	10 PIN SIL VERTICAL GOLD+TIN SOCKET	2	$1.49	$2.98
LM317DCYR	LM317	Linear Voltage Regulators3 Term Adj. Pos.	2	$0.74	$1.48
BSS138PS,115	BSS138	MOSFETN-CH 60 V 320 mA	2	$0.42	$0.84
CCF0710K0GKE36	10 Kohm resistor	Metal Film Resistors - Through Hole1/4watt 10Kohms 2%	4	$0.10	$0.40
ERG-2SJ241A	240 ohm resistor	Metal Oxide Resistors Metal Oxide Film Resistor 2W 5%	2	$0.56	$1.12
CB10LV102M	10 komh trimmer	Trimmer Resistors - Through Hole	2	$0.43	$0.86
				Total:	**$186.55**

Table 2: Bill of materials for the SMD version of Pinpoint

Part #	Name	Description	Qty	Unit Price	Cost
CY8CKIT-059	PSoC 5LP	Development Boards & Kits - ARMCY8CKIT-059 PSoC 5LP Dev Kit	1	$9.97	$9.97
1590	Touch screen driver	RA8875 Driver Board for 40-pin TFT Touch Displays	1	$34.95	$34.95
1596	Touch screen display	5.0" 40-pin TFT Display - 800x480 with Touchscreen	1	$39.95	$39.95
746	GPS receiver	Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates - Version 3	1	$39.95	$39.95
803063	Rechargeable Li-Po Battery	7.4V 1200mAh	1	$10.88	$10.88
N/A	Printed circuit board	FR-4 .062 1 Oz Cu, Up to 50 Square Inches	1	$30.00	$30.00
171-PA5525-1-E	DC Power Plug	2.5 mm power connector	1	$1.14	$1.14
161-0725-E	DC Power Socket	2.5 mm power socket	1	$1.14	$1.14
929834-02-36-RK	36 pin header stick	2.54 mm pitch header stick	2	$1.84	$3.68
M20-7820942	9 pin socket	9 PIN SIL VERTICAL SOCKET GOLD?	1	$1.20	$1.20
M20-7821546	15 pin socket	15 PIN SIL VERTICAL SOCKET TIN	1	$1.51	$1.51
929974-01-26-RK	26 pin socket	26 CON STR BRDMNT SKT	2	$2.25	$4.50
M22-7131042	10 pin socket	10 PIN SIL VERTICAL GOLD+TIN SOCKET	2	$1.49	$2.98
LM317DCYR	LM317	Linear Voltage Regulators3 Term Adj. Pos.	2	$0.74	$1.48
BSS138PS,115	BSS138	MOSFETN-CH 60 V 320 mA	2	$0.42	$0.84
3361P-1-102GLF	10 komh trimmer	Trimmer Resistors - SMD1/4" SQ 1K 10% 0.5WATTS	2	$1.32	$2.64
RR1220P-103-D	10 Kohm resistor	Thin Film Resistors - SMD1/10W 10Kohm 0.5% 25ppm	4	$0.02	$0.09
RR1220P-241-D	240 ohm resistor	Thin Film Resistors - SMD1/10W 240ohm 0.5% 25ppm	2	$0.02	$0.05
					Total: $186.95

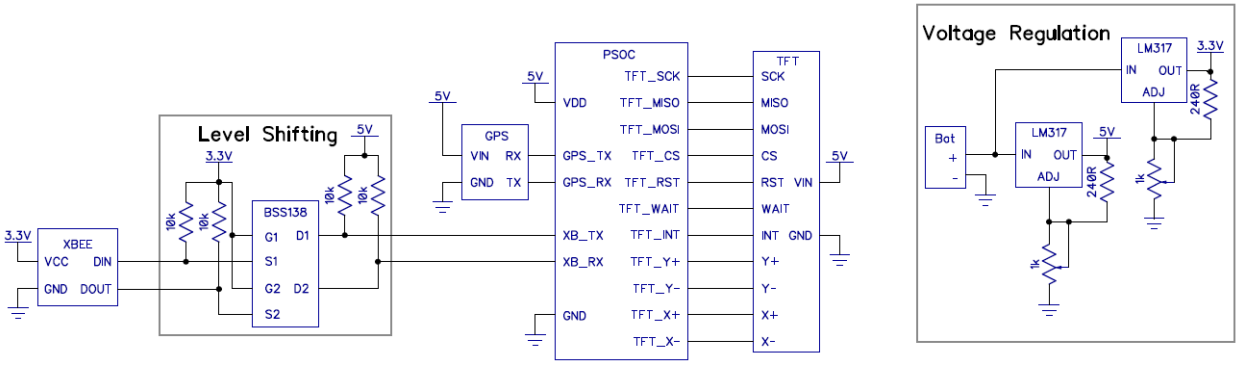

Figure 9: Hardware schematic for Pinpoint

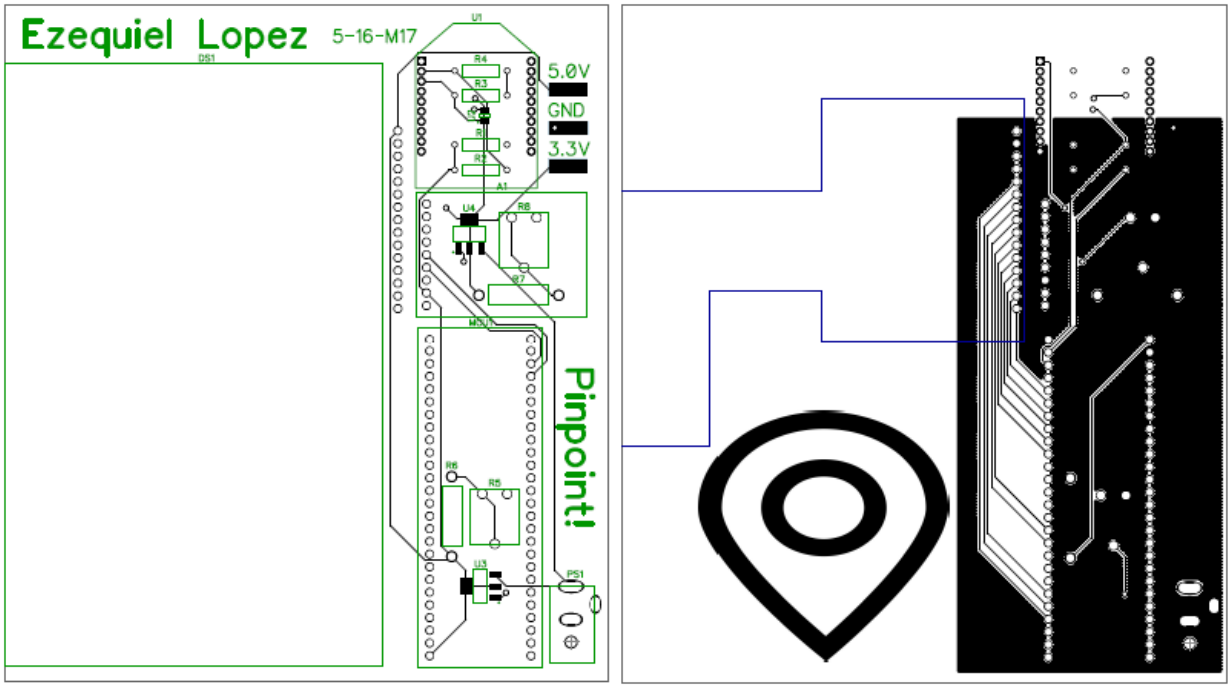

Figure 10: Through-hole PCB layout for Pinpoint

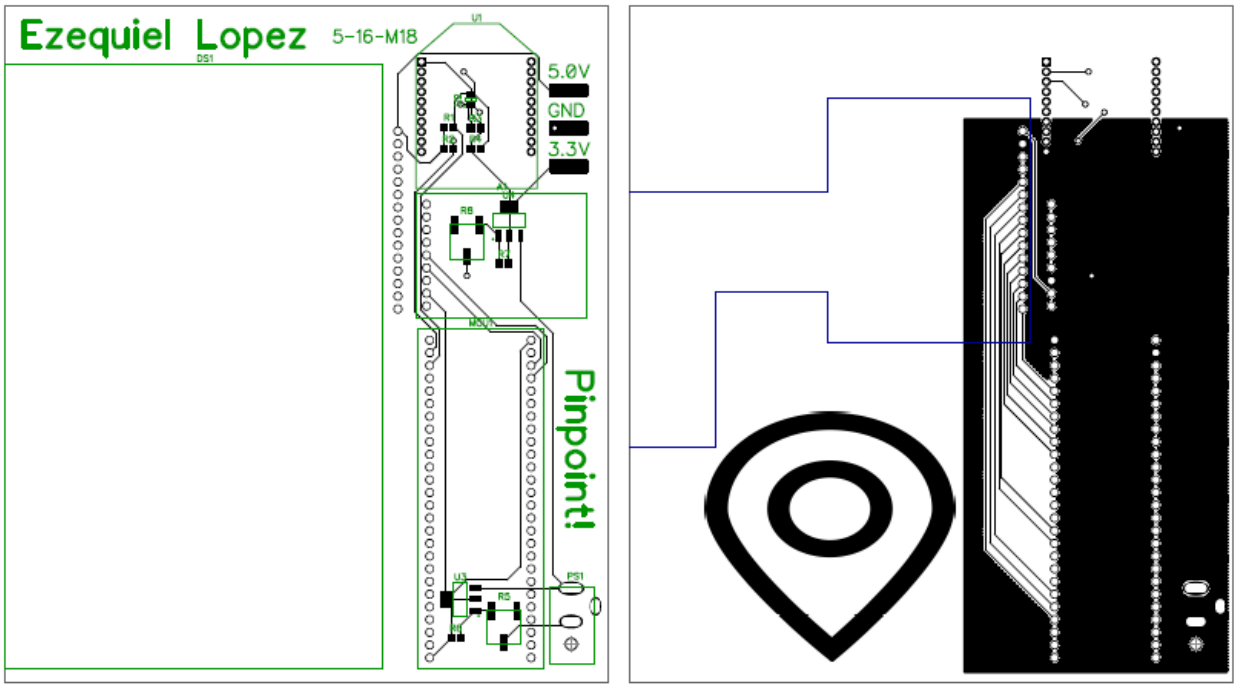

Figure 11: SMD PCB layout for Pinpoint