Statement of work:

Joshua Roy *(Computer Engineering)*

Joshua focused on the development of the Configuration Phone Application and developed all of the software to be run on the RedBear Nano V2 bluetooth modules. In addition, as a member of the team, he was also responsible for reviewing PCB layouts and circuit designs.

The Configuration Phone Application took up the majority of Joshua's time, as he designed the entire system with only a few pieces of relevant example files and templates, which he often ended up having to significantly modify. For example, while there were example files for connecting to Google Calendar from an Android device, Joshua had to convert the system from a subclass to a set of three classes (related by inheritance) to account for Google Calendar requests being called both automatically and at the user's request. He had to do significant research to understand automating event fetching, which included multiple background services, and researched bluetooth communication intricacies (even though bluetooth from the phone was not able to be integrated into the final system).

Joshua took a break in the middle of app development to develop software for the RedBear Nano V2 bluetooth modules, which first consisted of setting up SPI communication and then communication over Bluetooth. Joshua had to write SPI slave interfacing from scratch, and determined through testing the timing requirements that the master would have to adhere to in order to communicate with the slaves. He also adapted example sketches to send commands and text over bluetooth, and came up with the idea to and implemented the bed sensors' RedBear board to take the place of the bed sensors' MSP430 microcontroller.

Cassie Willis *(Computer Engineering)*

Cassie's role in the project was to develop the alarm clock printed circuit board, and to develop the code for the MSP430 microcontroller. In particular, Cassie researched and thoroughly analyzed the datasheets for the MSP430FR5962 microcontroller, the Riverdi display, the RedBear Nano V2 bluetooth modules, and other important datasheets to determine what the required circuit configurations for these components consisted of. She then combined these components onto one schematic design, and calculated any values needed for recommended circuit components, such as resistors for JTAG and capacitors for the crystal oscillator.

When transferring the schematic to a board layout, she made sure that all components were spaced appropriately as mentioned in the relevant datasheets, such as the Bluetooth modules having 10mm clearance on each side, and the crystal oscillator being placed as close the the MSP430 as possible. Cassie also ordered all of the components for this PCB and performed all of the assembly, with the exception of the MSP430, which was soldered by WWW Electronics. All testing for this board after construction was also performed by Cassie.

Finally, all of the code for the MSP430 microcontroller was written by Cassie. This code included information for setting alarms, turning alarms on, processing the sensor input data, handling snooze and person-in-bed functionality, and SPI communication with the phone

Bluetooth module. Cassie also acted as team manager, making sure parts were ordered, setting up team meetings, and communicating with the team to ensure everyone was working at a good pace and completing tasks.

Samuel Byers *(Electrical Engineering)*

Sam's role in this project was to develop and design the bed sensors and relevant circuit boards. This included researching and determining the best sensors to use, designing and drafting a system by which the sensors could communicate with the alarm clock, as well as board assembly. Testing of these boards was also performed by Sam. In addition to this, Sam was also in charge of designing, modelling, and printing cases and plastic parts in which to house the boards and other components.

Table of Contents

Contents

Table of Figures

Abstract

The Internet of Things (IoT) Alarm Clock is a project that solves a problem that a large portion of the world's population faces every morning - waking up. The clock was designed to include features that will make waking and staying up easier than ever. The clock includes pressure sensors which are placed under the user's bed, which send signals to the clock if the user turns their alarm off and remains in bed. If the clock determines the user is still in bed, the alarm will be turned back on. The alarm also connects via a phone application to the user's calendar, automatically setting alarms based on the first daily event in the user's schedule. This project made use of Bluetooth communication and the MSP430 family of microcontrollers among other components, and societal, environmental, and economical factors were considered along with the technical aspects of the project.

Background

The idea of an IoT alarm clock came out of a personal problem to which team member Cassie desperately needed a solution. Cassie found in her own life that waking up on time in the morning is very difficult. She often finds herself late to classes or work because of her bad habit of hitting the snooze button five times before getting out of bed, or accidentally turning the alarm completely off. Figuring that this was a problem that many other people also shared, the idea of the IoT alarm clock was born. The original project began only as a way to automatically set and change alarms based on the user's calendar and email accounts. The project then evolved to include the bed sensors, which would solve the problem of people turning their alarms completely off and going back to sleep. The email parsing portion of this project was eventually removed do to time and resource constraints.

The Internet of Things was chosen in particular for this project because of its booming presence in the global market and in the technological community. The possibilities for connecting previously separated electronics to each other and to the internet allows a whole new experience for the user [1]. IoT is so large that GE is expecting the industry to top $60 trillion in the next 15 years [2]. IoT can be used for entertainment, security, automation, and health purposes. With the growth and opportunities IoT brings, delving into an IoT project will provide invaluable experience when entering the technology industry upon graduation [1].

This project was based on the idea of an IoT smart mirror, built two years ago as a capstone project, but originating as a personal project by Michael Teeuw three years ago [3]. Since launching, the Magic Mirror project has swept the globe, with copycat projects appearing in numerous countries and, as stated, even here at the University of Virginia. Enthusiasts, including Michael Teeuw, have been adding modifications and creating an ever more complex system. The project has interested one of the group members since 2015, however, its significant following and the multitude of resources available for it have made the project less challenging over time, and too simple for a capstone project. The alarm clock project is a twist on this idea

which uses an MSP microcontroller instead of a Raspberry Pi and tackles some different features, making it more of an engineering challenge [1].

IoT alarm clocks do currently exist in the market, the most notable being the Vobot LED Display Smart Radio Alarm Clock Speaker and the Widdi Beddi Style. These devices tend to focus on the "extra" features, however, more so than the alarm clock, which appears to not be much smarter than an alarm clock without IoT connectivity. For example, the Vobot clock uses a connection with Amazon Alexa to perform all of its IoT functionality, and the Widdi clock's phone application is primarily for alarm settings, display brightness, and playing music [4], [5]. The project proposed here takes a different approach to IoT, using data mining of Google calendar services to change alarms automatically, and adding a bed sensor to sense when the user has ignored their alarm and gone back to bed.

It needs to be noted that the Bonjour Smart Alarm Clock already consists of many of the features that are proposed here [6]. However, it is possible that the Bonjour clock is a scam product, as research showed multiple crowdsourcing sites and contradicting delivery days for the device, which is still not available to the general public. Even still, this project's bed sensor adds functionality that the supposed Bonjour clock is missing.

This project brings in a lot of the concepts taught in the Computer and Electrical Engineering coursework seen so far. This includes coding learned in the required class sequence in the Computer Science department, embedded programming taught in the Introduction to Embedded Systems course, and circuitry design and development taught in the Fundamentals series. One of the group members also took a course in Cyber Physical Systems, and one in Advanced Embedded Systems, both of which proved somewhat useful for this project [1].

Constraints

Design Constraints

The largest constraints on this project were time and money. The system had to be designed to be completed in one semester by a three-person team, with a budget of no more than $300. The time constraint ended up being the most restrictive, as numerous desired features had to be acknowledged as beyond the scope of a one-semester project.

There were some implicit design constraints that had to be considered. First of all, alarm clocks are small and portable. The design was implicitly constrained by that fact, and by the Major Design Experience requirements, so that using a laptop was out of the question; the team would have to use a microcontroller to handle generating the alarms. Similarly, the user's idea of an alarm clock restricted the user interface of the clock. Alarm clocks simply don't have keyboards or complicated interfaces, so the design had to include a separate user interface - in this case, the development team chose the user's mobile phone.

The development tools used by the team were constrained by their choices of technology. Android Studio, for example, one of few IDEs that can compile and debug Android applications. Similarly, programming for the MSP and RedBearLab microcontrollers were constrained by the

existence of only a few officially supported development environments. The majority of the design tools were constrained by what tools were reasonable for the team to learn and operate in the course of their single semester of work.

While in some cases standards did not directly influence or constrain the design, the constraints of circuit board printing led to constraints on the design of said circuit boards. Because our manufacturing partner required compliance with these standards, our printed boards were not able to be laid out in an optimal fashion. The Major Design Experience requirement of industrial-level connections proved to be the greatest constraint on the design of our printed circuit boards, as we had to rely on direct connections without using jumper wires or breadboards.

Economic and Cost Constraints

The budgetary constraint did not explicitly constrain the design of the project. Because our project was significantly under budget, we had the freedom to request fast shipping and last-minute printed circuit boards without becoming a hindrance to the department. The main way that we decreased our cost is that we made use of the user's mobile phone instead of including the phone as part of our system - if we hadn't, the cost of a mobile phone would've added multiple hundreds of dollars to the cost of the project.

External Standards

The large amount of both wireless and physical communication between devices in this project led to the use of many different industry standards, though most of them were not dealt with directly. The process of fetching events from Google Calendar required the standards for WIFI (IEEE 802.11[7]) and HTTP (HTTP/1.1[8]), though the implementation of those standards was accomplished by provided libraries. Similarly, the JTAG interface (IEEE 1149.1-1990[9]) was handled by a JTAG interfacing device and technology already built into the MSP430 microcontroller. Finally, the USB standard for data communication (IEC 62680-1-3[10]), which was used to program the RedBearLab chips, was handled by the laptop and RedBearLabs programmer.

In the end, there were four protocols that had to be directly handled by the developers. The first is UART communication, outside of setting standard baud rates, it was also handled largely by external libraries. This communication was especially useful during troubleshooting of the RedBearLabs software, and the baud rates actually did play a large part in the debugging process. If too small of a baud rate was used, the communication would alter the timing of the microcontroller's signals, so the baud rate was increased to 1,000,000 bits per second. The second standard that was implemented was Serial Peripheral Interface (SPI), which, while not formalized by any professional organization, has a standard implementation. This protocol actually had to be written to specification by the developers, and the slave code developed for the project is compatible with other SPI masters. The only formalized communication protocol that had to be significantly implemented was Bluetooth (formerly IEEE 802.15.1, now formalized by

the Bluetooth SIG[11]). Bluetooth Low Energy is a large part of the project, and developing in accordance with its forms of communication proved to be a significant amount of work.

Finally, there was one main standard that had to be followed by the team during PCB design. IPC-7351C defines requirements for part spacing in through-hole components on printed circuit boards, and those requirements had to be met for the boards to be printed [12].

Tools Employed

The primary software tools employed in this project were Android Studio, Arduino, and Code Composer Studio. Android Studio was used to write, compile, and test the Java programs that made up the Configuration Phone Application. Arduino was similarly used to write, compile, and test the code for the RedBearLab boards. While it is not the only option to program those boards, Joshua was already familiar with using the Arduino IDE and writing Wire code, so it was a natural choice to program the RedBearLab boards. Code Composer Studio was used to write the C code for the alarm clock's MSP430. In all of these cases, the programmer already had experience using each IDE, but that experience was expanded upon, either by using new libraries and features or building code for a new board.

National Instruments' Multisim, Ultiboard, and VirtualBench were all used in the design and testing of the electrical systems in this project. Multisim was used to design and test the circuits in simulation, Ultiboard was used to lay the components onto a simulated printed circuit board, and a VirtualBench was used to test the final products. The VirtualBench was also used to test the SPI communication between the RedBearLab boards. The team was already familiar with all three of these programs.

Ethical, Social, and Economic Concerns

Environmental Impact

The IoT Alarm Clock was determined to not impact the environment in a significant way. The only consideration to be understood is the environmental costs of production and operation, which only consists of the building of the PCBs and 3d printed components and the generation of the electricity to power the system. The bed sensors do require the use of a battery, and thus it must be disposed of in an environmentally safe way. There is no way to know how the system's ability to wake up users at the proper time will affect the environment.

Sustainability

The IoT Alarm Clock is a very sustainable system in many ways. First of all, the components themselves are extremely durable, as the product does not contain moving parts and is housed in a container that can withstand damage from expected use. The industrial-level connections within the system ensure that the connections will not quickly wear out, with the only wire-based connections being the power cable and the ribbon cable connecting the display to the alarm clock PCB. The most likely sources of permanent system failure are an update to

either the Android API or phone technology that will render the Configuration Phone Application unusable or the LEDs on the display wearing out, though neither of those events are expected to occur in the foreseeable future.

As a product, the IoT Alarm Clock is also seen as sustainable. Alarm clocks as a technology have been used since the 1200s [13], and are not expected to be rendered obsolete any time soon. It is possible, though, that another company will produce a new alarm clock that introduces more features and inadvertently causes the decline of use of the IoT Alarm Clock.

Health and Safety

The IoT Alarm Clock was designed with health and safety in mind, resulting in it being a very safe system to own and operate. The system has very low power requirements, with only enough power to drive the microcontrollers required by the system for normal operation. The first step after introducing the 120V external power to the alarm clock is to step it down to 3V, which is much safer, and even that voltage (as well as the 9V power of the Bed Sensors) is contained within a plastic housing. The bed sensors themselves represent the most significant health hazard, but only if the casing of the wires wears down, and even then it will not result in a short circuit, due to the high input impedance of the unity gain buffer and the 10KΩ resistor.

Manufacturability

The IoT Alarm Clock is highly manufacturable, though certain changes would be made if that were the case. As of right now, the IoT Alarm Clock consists mainly of solder connections, and those can be manufactured, but even the through-hole connections for the basic circuit elements (ex. resistors, capacitors) would be replaced by surface-mount packages. Similarly, the RedBearLab boards may also be modified to allow them to be soldered directly onto their PCBs, instead of the current system of plugging them into stackable headers. The final change that would likely be made is the use of molds or another cheaper alternative for 3D printing for the production of the casing.

The largest change that would actually have to happen to enable mass manufacturing of the IoT Alarm Clock is changes to the software. As of right now, the Bluetooth boards look for a specific name of a device to connect to it (“IDK_Bed_Sensors”), so if multiple clocks were used within range of each other, the clocks could connect to the other set of bed sensors. The main change that needs to be implemented is either to pair clocks and sensors with unique identifiers or to use a “sync” button on both the clock and the sensors to allow the user to connect their clock to the correct bed sensor system.

Ethical Issues

Ethical issues were not considered as a part of the development of this project, because it is generally considered that waking up on time is morally neutral. The main ethical concern is then what users do with their time, but those considerations are beyond the realm of possibility

for this project. It is possible that, by linking multiple IoT Alarm Clocks to the same Google Calendar, that nefarious users could use the system as an excellent tool for synchronizing activities. It is possible that there is no benign reason to have multiple clocks connected to the same account, and in that case, that restriction would be put in place to prevent clandestine operatives from performing malign actions with multiple clocks.

Intellectual Property Issues

The claims for the system would be

1. An alarm clock
2. Claim 1, that automatically pulls events from a calendar service
3. Claim 1, that is connected to weight sensors in the user's bed
4. Claim 3, wherein the clock will turn itself back on if the user is in bed

The connection to the mobile phone is actually a subsystem that would not be claimed in the patenting of the system, because the novelty of the system does not require the phone as a user interface. The claims listed include a similar system that uses a wifi-enabled microcontroller and a keyboard for user- and internet interfacing.

The system would likely not be patentable, as each component is already covered under a patent, and thus the integration of the subsystems would be obvious. Portions of this system that are already patented are:

- Setting the snooze interval in a mobile phone application [14]
- Using a smart device to set the alarm [15]
- An alarm clock that senses an occupant (even so far as suggesting weight sensors) [16]
- A separate "alerting unit" issuing an alarm to a local terminal [17]

The one portion of the system that could even be thought to be patentable is Claim 2, and even then, alarms are a feature that are usually bundled into calendar services already. It could then be reasoned that the system behaves identically to a calendar service that outputs alarms on a separate device, and smart devices setting off alarms on other devices is likewise already patented.

Detailed Technical Description of Project

Overview

The IoT alarm clock was developed as three subsystems working parallel to each other and communicating over Bluetooth. Hardware designs were created for the alarm clock board as well as for the bed sensor board. Software was developed for the alarm clock MSP430 microcontroller, the Redbear Nano V2 Bluetooth modules, and the phone application. Figure 1 shows a breakdown of the overall system design.

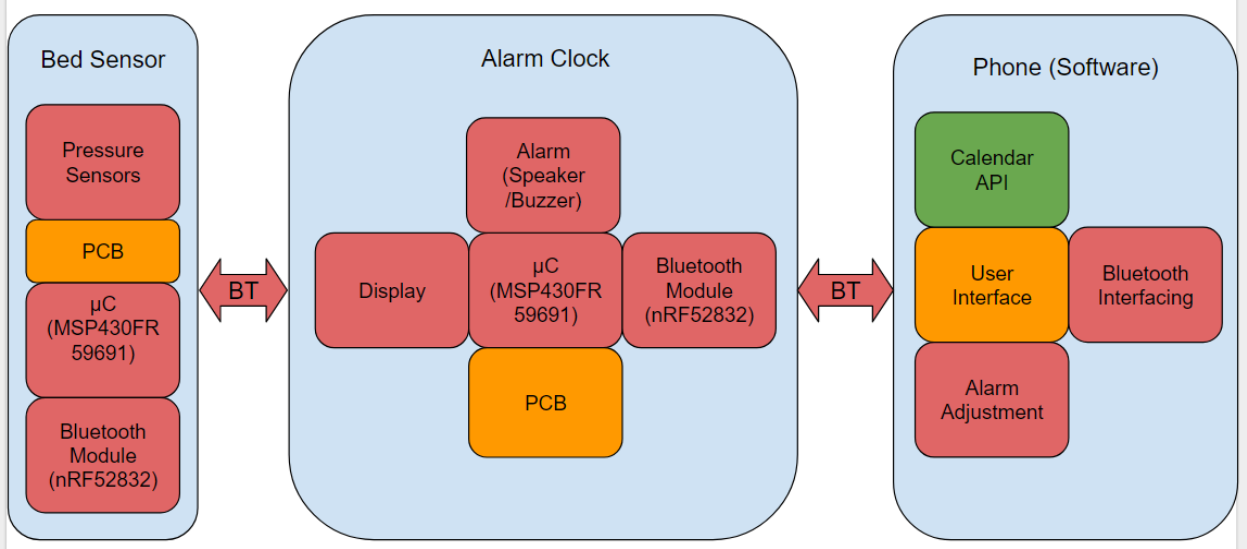


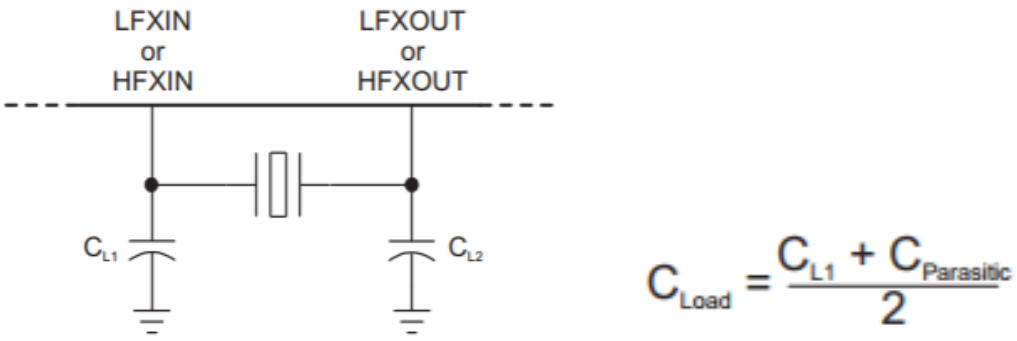
Figure 1: Overall system design

Hardware Design

Alarm Clock PCB Hardware Design

The alarm clock PCB is the central hub for the IoT alarm clock project. It is composed of several subsystems, including the LCD display, two Bluetooth modules, an MSP430 microcontroller, JTAG interface, switches, power supply regulation, and MSP430 peripheral circuits.

The MSP430 microcontroller is the base for which the rest of the alarm clock board is centered around. The microcontroller includes power supply decoupling for both AC and DC power supply options. This decoupling exists in the form of capacitors placed in parallel to the AC and DC power supply pins. External voltage decoupling is also included for decoupling a voltage supplied by the microcontroller to the switches on the board. This decoupling is also comprised of two parallel capacitors. A crystal oscillator was included on the board to provide an accurate frequency for the real-time clock on the MSP430. The schematic and calculations needed for the suggested operating conditions of the crystal are shown below. Because a high frequency was desired, the crystal was connected to the HFXIN and HFXOUT pins. The capacitor values for the crystal circuit were chosen to be 0 pF due to recommended values for the MSP430F* family of microcontrollers being 0 pF to 12 pF.



Figures 2A, 2B: Crystal Oscillator recommended circuit and capacitor formula

The final component that had a recommended peripheral circuit for the MSP430 was the JTAG interface. The recommended circuit for the JTAG is shown below, and the J1 connection was chosen because the designers chose to power the JTAG through the board power and not the JTAG device power so that a constant voltage could be ensured.

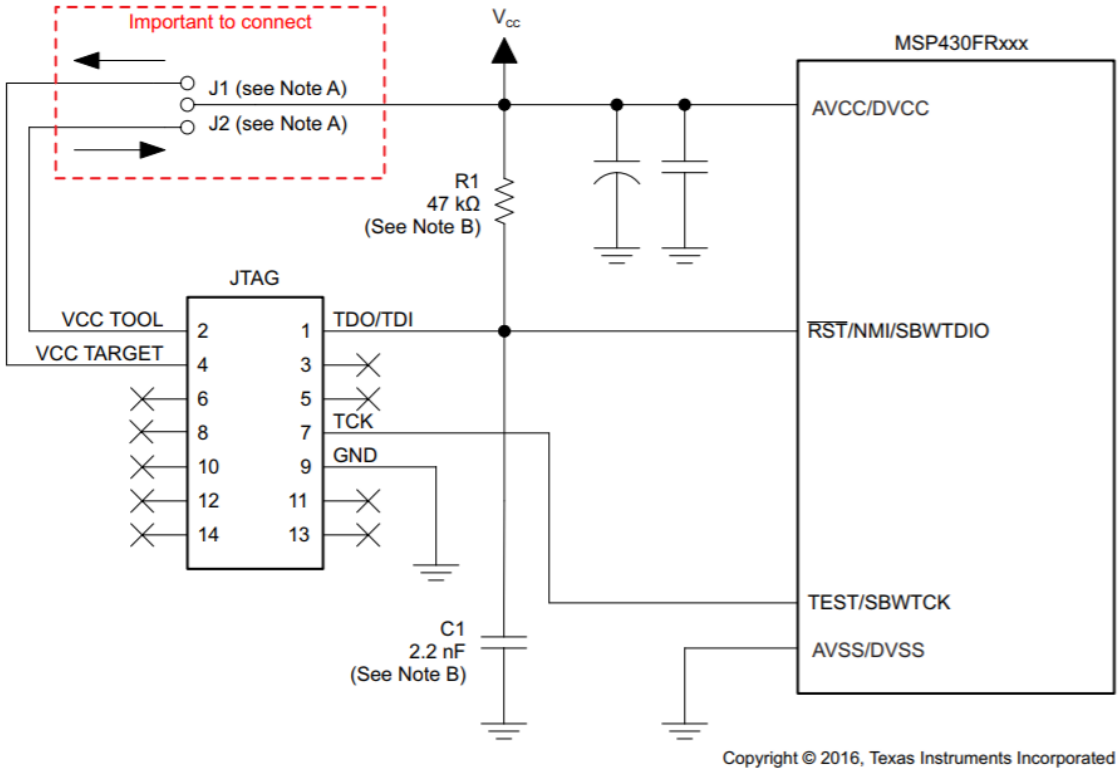


Figure 3: JTAG recommended circuit

The designers wanted the alarm clock to be powered using a wall outlet because a wall outlet does not have the problems of running out of charge, and also because of the convention of most alarm clocks currently on the market. Because of this, a voltage regulator circuit had to be used. Resistor and capacitor values were chosen as seen below to output 2.7V to the board.

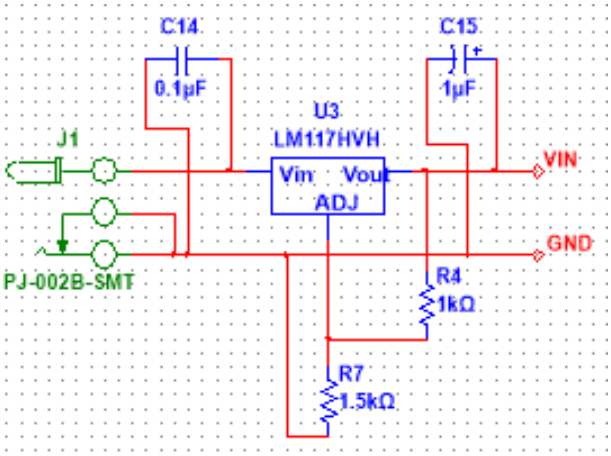


Figure 4: Voltage regulator circuit

A piezo buzzer was chosen as the alarm sound device for the alarm clock. An NPN bipolar-junction transistor and a potentiometer were added to the buzzer subcircuit to control when it was turned on and volume. The subcircuit is shown below.

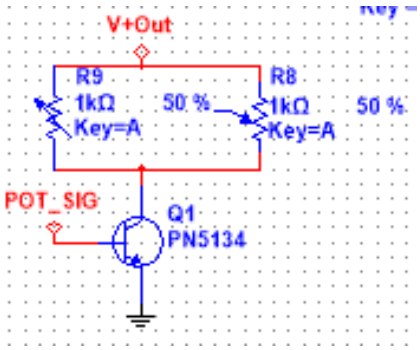


Figure 5: Piezo buzzer subcircuit

The switches chosen were single-pole-single-throw switches connected to the power supply and then to the MSP430. If the switch is thrown, the voltage signal will be sent to the microcontroller, signaling to change whatever value the switch was meant for. Switches were created for snooze, power on, Bluetooth pairing, and turning the alarm off. Finally, the LCD display and Bluetooth modules were both connected via SPI to the MSP430, and used connectors such as header pins and a ZIF 20-pin connector. Pin connections were determined through appropriate datasheets.

Bed Sensor PCB Hardware Design

The section furthermore referred to as the "Bed Sensors" is composed of a series of force-sensitive pads which vary in resistance according the amount of pressure applied. The voltage

through each sensor is then sent through a unity gain buffer (for stability) and the transmitted via Bluetooth to the main alarm clock board. The purpose of this portion of the device is to detect whether or not the user is currently in bed. This is done by placing the force-sensitive sensors beneath the user's sheets and analyzing the values being sent over Bluetooth. Due to how the sensors change resistivity, a high voltage signal indicates that pressure (i.e. the user) is present while a low voltage signal indicates that the sensors, and thus the bed, are unoccupied.

The bed sensor system is powered by two AA sized batteries held within a standard battery case. The circuit then splits into four paths, one for each sensor (*Figure 6: Bed sensor block diagram*). After passing through the sensors, the next stage of the circuit is a unity gain buffer, composed of a LMC6482IN Operational Amplifier and a 10k Ω resistor. (While each sensor has its own dedicated resistor, each pair of sensors shares a physical op amp). Each op amp also has a 0.1 microfarad bypass capacitor. After the signals are stabilized with the unity gain buffers, each signal is sent to a dedicated port on the RedBearLab chip, which then begins transmitting the signals over Bluetooth.

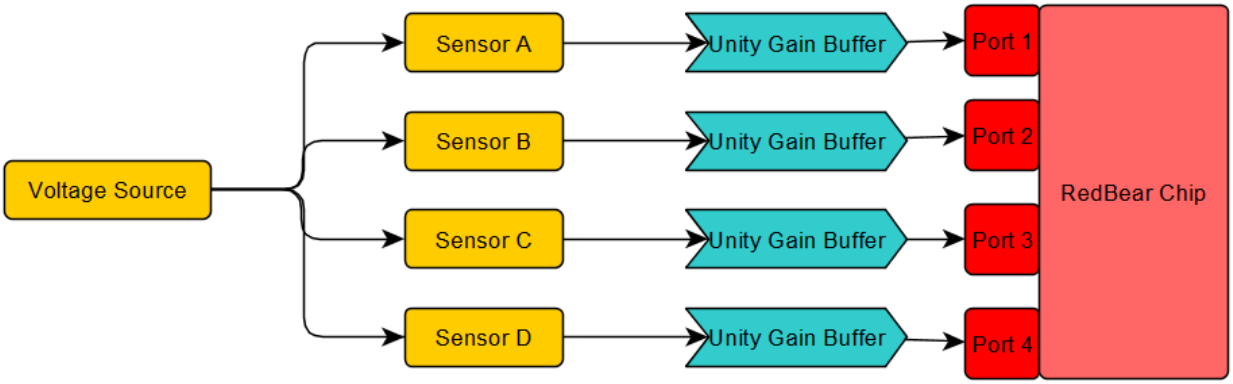
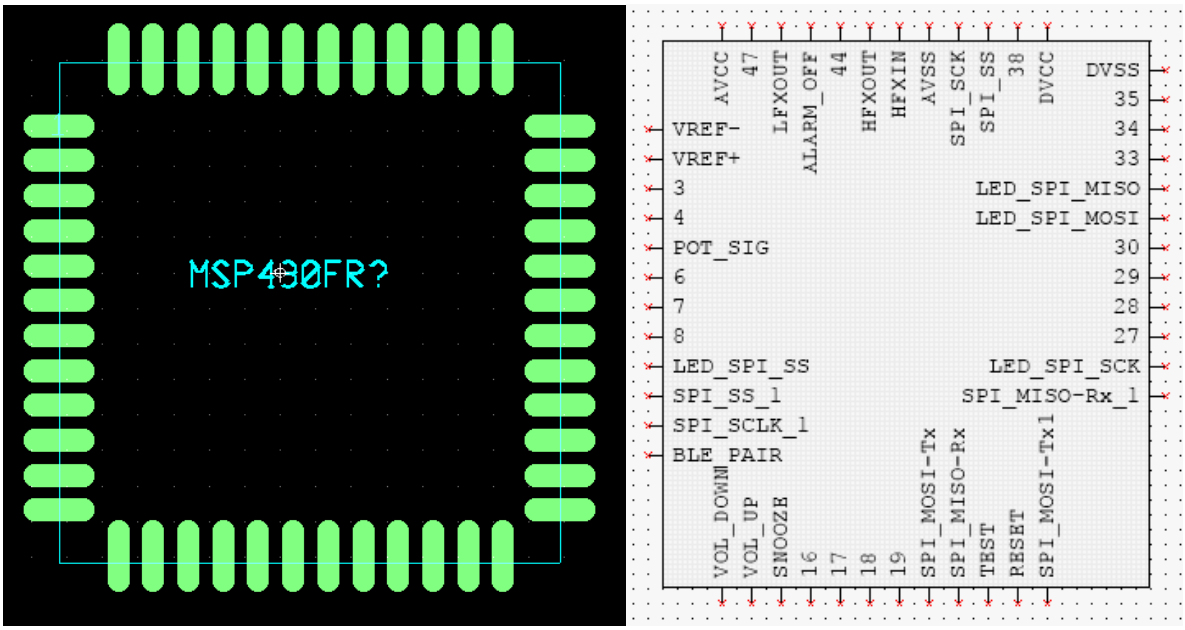


Figure 6: Bed sensor block diagram

Ultiboard Part Creation

Several components of the two PCBs were parts that are not included in the standard libraries for Multisim and Ultiboard. These included the MSP430FR5962 microcontroller and the Redbear Nano V2 Bluetooth module. Though a library component was created for the Bluetooth module, the final design used headers in place of directly soldering on the Bluetooth module, so that that modules could be removed for programming. The MSP430 component was used in the final design, as the chip was soldered directly onto the alarm clock PCB for space saving purposes.

To create the custom library part, the datasheet for the MSP430 was consulted. The Ultiboard footprint was created via the Part Wizard tool. The schematic part was then designed in Multisim, and the pins were named appropriately and mapped the Ultiboard footprint. Figures 7A and 7B show the custom library component.



Figures 7A and 7B: MSP430FR5962 Library Component - Footprint and Schematic

Alarm Clock PCB Schematic and Board Layout

The schematic, as described above, was primarily determined through the various datasheets for recommended schematics for the various components on the board. The schematic was laid out and labeled according to subsystems, and can be seen below.

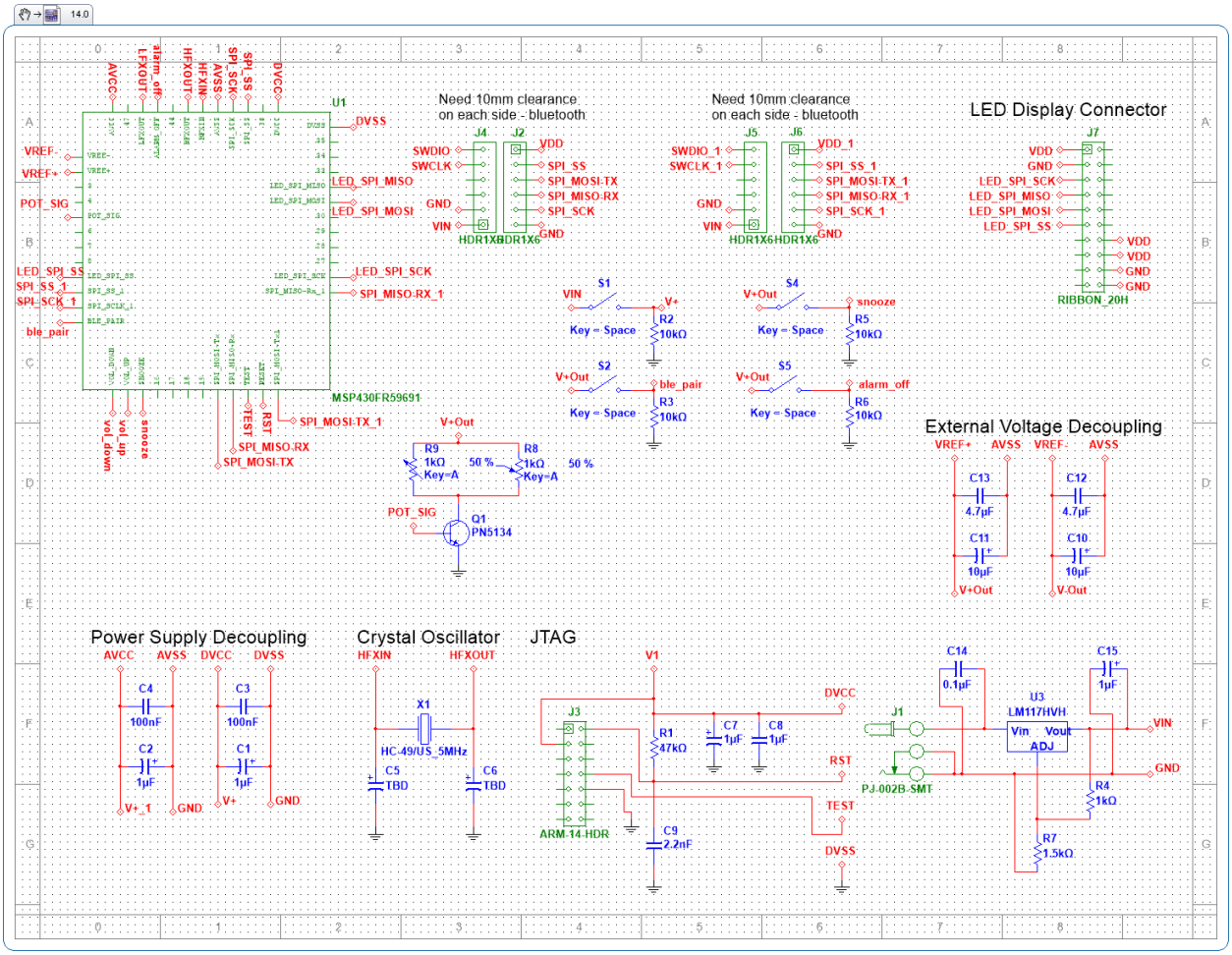


Figure 8: Alarm clock schematic

The board layout was also chosen somewhat in relation to suggestions on the datasheets, and somewhat from lessons learned in previous Fundamentals classes. The resistors and ceramic capacitors were lined up together for tidiness and due to them sharing common ground. The electrolytic capacitors were also put together in square shapes that allowed for shorter traces, and the switches were placed on the board similarly. The MSP430 was centered onto the board since it had the largest number of traces connecting to it so more room was required to route those traces. The crystal was placed close to the MSP430 so that the frequency would not be distorted in the trace from noise. All connectors were placed on the edge of the board so that the wires were easy to connect to the connectors. The board layout can be seen below.

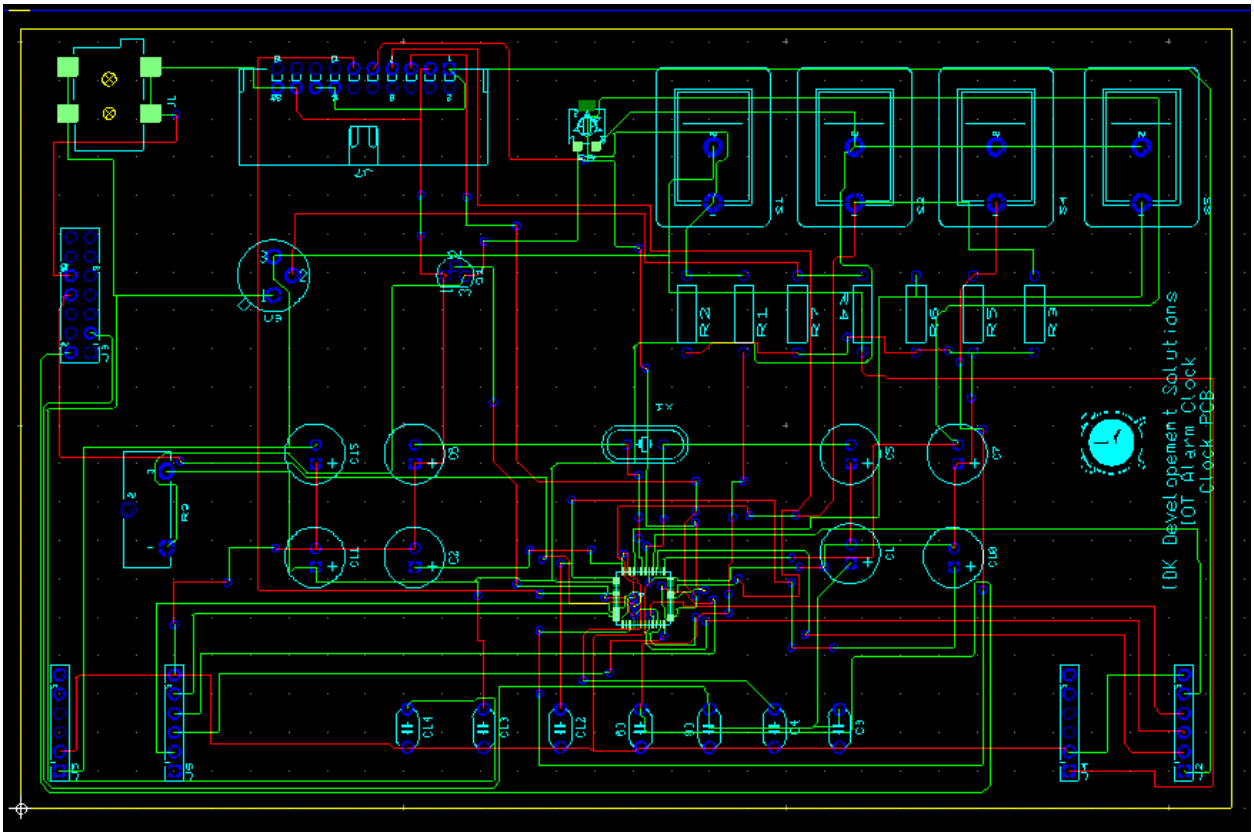


Figure 9: Alarm clock board layout

Bed Sensor PCB Schematic and Board Layout

The design for the circuit board containing this portion went through several design phases. Initially, the plan called for an integrated MSP, which would assess the value of each signal, determine if the user was triggering the sensors, and then send only a single binary signal to the Bluetooth chip stating if the "bed" was occupied or not. To accommodate this, the original plan was to use a 9V battery and have voltage divider circuit step the voltage down to the 3.3 volts needed to power the MSP430. However, due to a misunderstanding regarding how the MSP430 worked, the board was designed in such a way that it required a direct wire connection to an external MSP430. The board was then redesigned so that the unity gain buffers would send their signal to the RedBearLab chip directly (*Figure 10: Bed sensor schematic*). While this did require us to rapidly redesign, order, ship, and test the board, it also opened the door for two potential gains. Firstly, the burden of calculating to determine if the user was on the sensors was offloaded to the MSP430 on the main alarm clock board. The code for the process was easily melded into the existing code on the alarm clock MSP430, and the need and expense of a second one was obviated. Secondly, removing the MSP430 from the bed sensor circuit allowed us to reduce the power supply from nine volts to three. This made it a) easier on potential users to fix the power supply when it depletes b) safer to use due to lower voltage c) less of a strain on the RedBear chip which was not designed to transmit signals above a low voltage threshold.

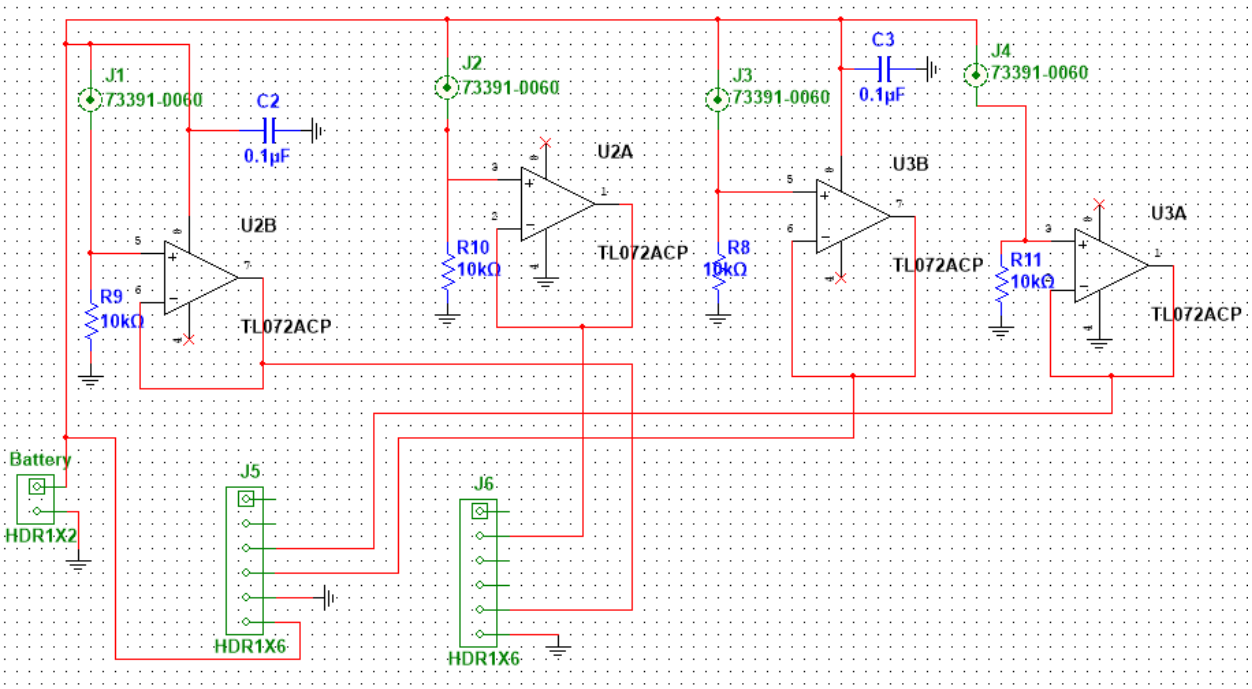


Figure 10: Bed sensor schematic

Another change implemented from the original design was the method of connection between the sensors themselves and the boards. Originally, we intended to connect the sensors via miniature coaxial cables. The boards were designed and printed with the five-hole structure needed to accommodate the mounts (*Figure 11: Bed sensor board layout*). However, due to time constraints and purchasing difficulties, the cables themselves were not able to be acquired within the allowed timeframe. As a workaround, two lengths of insulated single strand copper wire were directly soldered between each sensor and the section where the coaxial cables should have connected. While not the most practical workaround, some sacrifices had to be made in the interest of time.

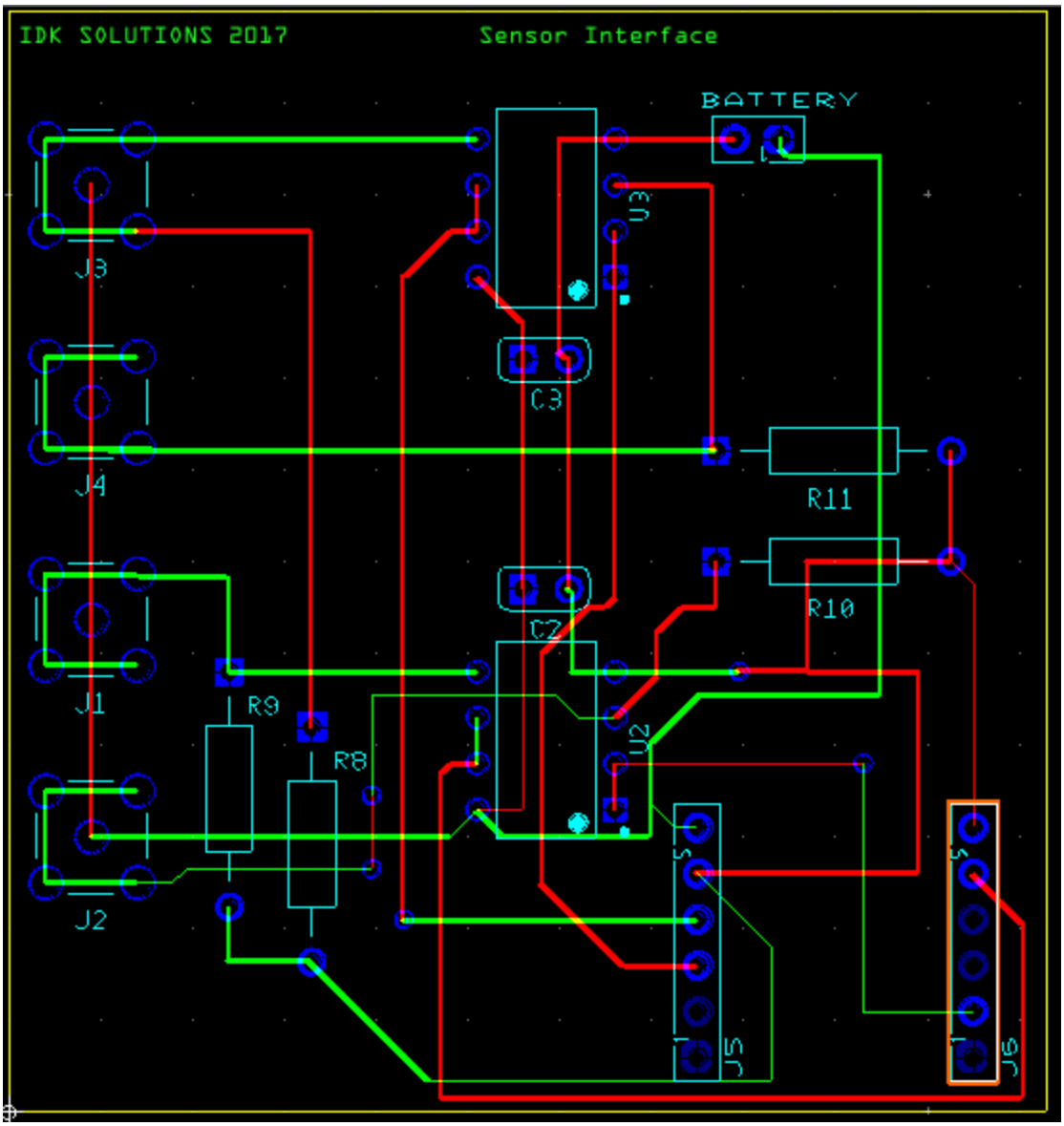


Figure 11: Bed sensor board layout

Software Design

Phone Application Software

The Configuration Phone Application is the channel that the Alarm Clock uses to gather information from the outside world and input from the user. This information is mainly transmitted as the alarms that the system needs to run, which are created by the app in response to the user's preferences and schedule, as hosted on Google Calendar. After creating a set of alarms, the Configuration Phone Application transmits the times of alarms and text of any notifications to the Alarm Clock via the Bluetooth communication protocol.

The application was implemented through two Android activities, as well as seven additional classes. The most important class is MainActivity, which contains the majority of the user interface and handles calling all other classes. In this prototype, it has a very simple user interface (*Figure 15: Picture of phone screen*), consisting of four buttons and an output space. Pressing the first button causes the system to generate alarms for the next week by instantiating a MakeCalendarRequestTaskActivity object. The second calls the email api to attempt to find any recent email listing class cancellation and crudely puts the information onto the screen. The third button transitions the system to SettingsActivity, which allows the user to change the name, alarm lead time, and snooze time that the application uses (*see Figure 18: Settings Menu*). Finally, the fourth button toggles whether or not the system automatically checks for new events every minute (see *Figure 14: Automatic Alarm Generation*)

Creating the list of alarms is a fairly simple algorithm (*Figure 13: MakeCalendarRequestTask Block Diagram*). First, the system creates a request to Google Calendar's API to respond with all of the events for the next week. Then, the system selects the first event from each day, only including the current day if no event has already occurred. Next, it subtracts the alarm lead time from each event. Finally, it checks whether those events are the same as those last save or not. If they are not the same, then the system saves the new alarms to the phone's memory and sets the flag to show that the file contains new data. If they are the same as the previous saved data, then no file is modified.

All of this functionality is contained within the MakeCalendarRequestTask activity, but it is called by creating one of its children, either MakeCalendarRequestTaskActivity or MakeCalendarRequestTaskService (hereafter referred to as MCRTA and MCRTS, respectively; also *see Figure 12: MakeCalendarRequestTask Inheritance Tree*), depending on whether the system is generating events through MainActivity's button (MCRTA) or through automatic alarm generation (MCRTS). The primary difference between the two is what is done with the data at the end of the process; in addition to potentially saving the data, MCRTA outputs the alarm list to the screen for the user to see (*see Figure 17: Generated Alarms*) and MCRTS sends the list to the debug log because it cannot output to the user interface.

Integration with the mail API was designed with the intention of developing a similar three-class structure, but because development on it stopped before it could be completed, it only has the base and Activity classes, with no Service class. Its operation is even simpler than that of the calendar request; it simply gets all emails that are from Megan Lowe, the University employee responsible for sending out class cancellation notices, that include the words "cancelled" or "resume", and outputs those messages to the screen (*see Figure 16: Email output*).

The final major component to the Configuration Phone Application's class structure are the two classes responsible for handling the automatic alarm generation (*see Figure 14: Automatic Alarm Generation Block Diagram*). Alarmer is a BroadcastReceiver class that, whenever it receives an Intent, tells FetchingService to run its method handleActionFetch(). This

method is the primary code block of the FetchingService class, and handles creating a new MCRTS and printing the result to the debug log.

The primary challenges in developing the Configuration Phone Application were in learning how to implement the desired system in Android. The primary example of this is the implementation of MakeCalendarRequestTask. Originally, the functionality of that class was written as a subclass of MainActivity, because that is the way that the example code from Google was written. The process of expanding the functionality of the example into FetchingService took far longer than expected because the process of using BroadcastReceivers and IntentServices had to be learned during development. The three-class model was developed to use the same base code whether or not the caller has access to the user interface.

Another challenge that had to be scoped out of the product was the implementation of detecting class cancellations through email scanning. While this was a cool idea, the difficulty in understanding the difference between a class and assignment cancellation was not given proper consideration at the beginning of design. Taking a real-life view at the feature, there truly was no room for error, and since the system could not be guaranteed to work, it was removed from the final product.

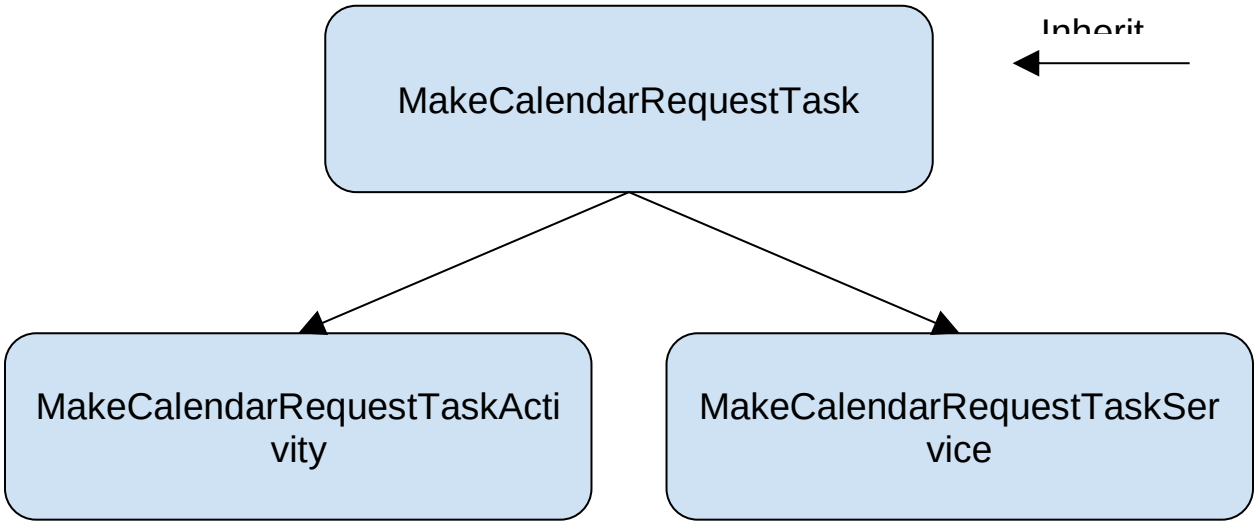


Figure 12: MakeCalendarRequestTask Inheritance Tree

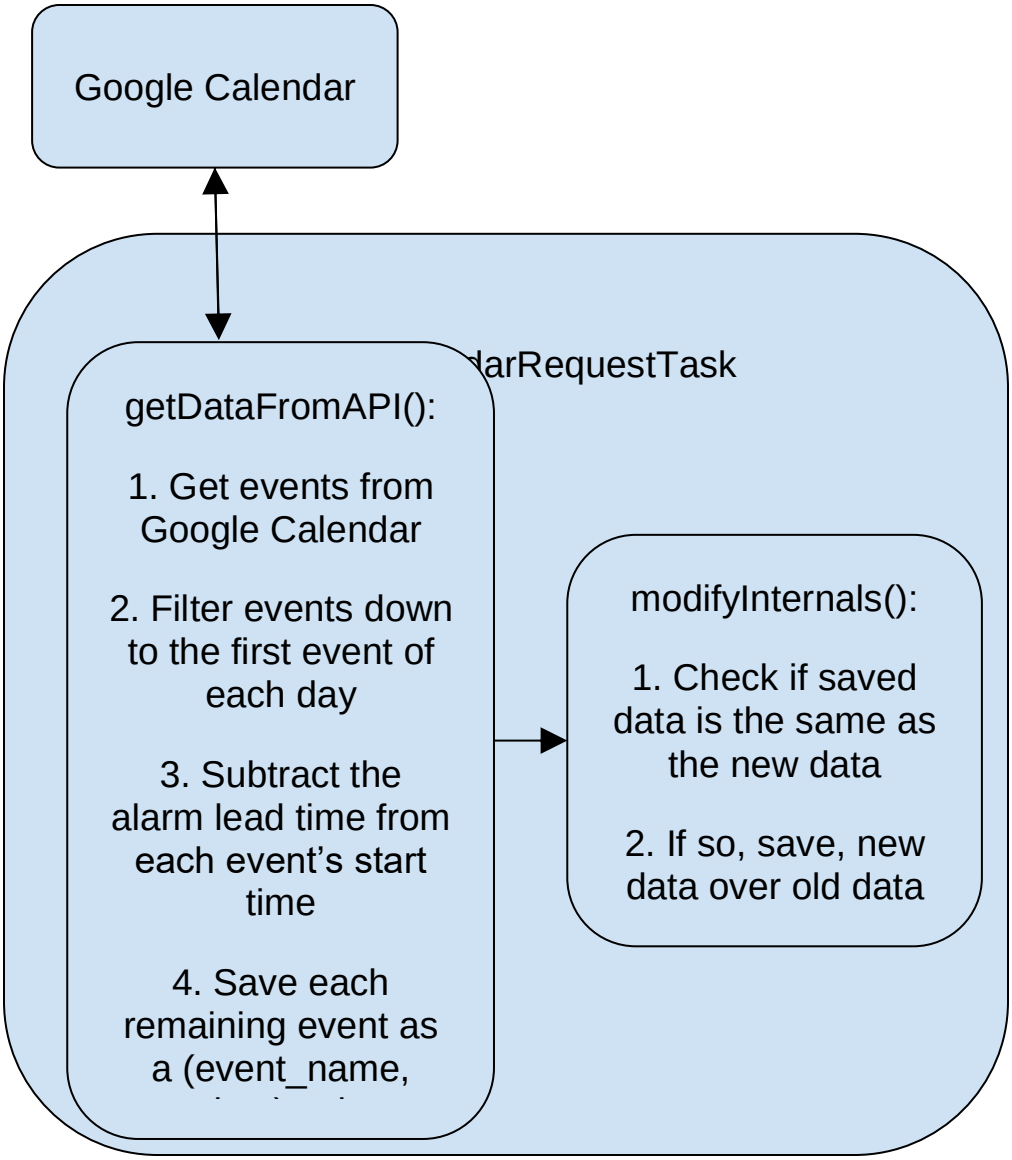


Figure 13: MakeCalendarRequestTask Block Diagram

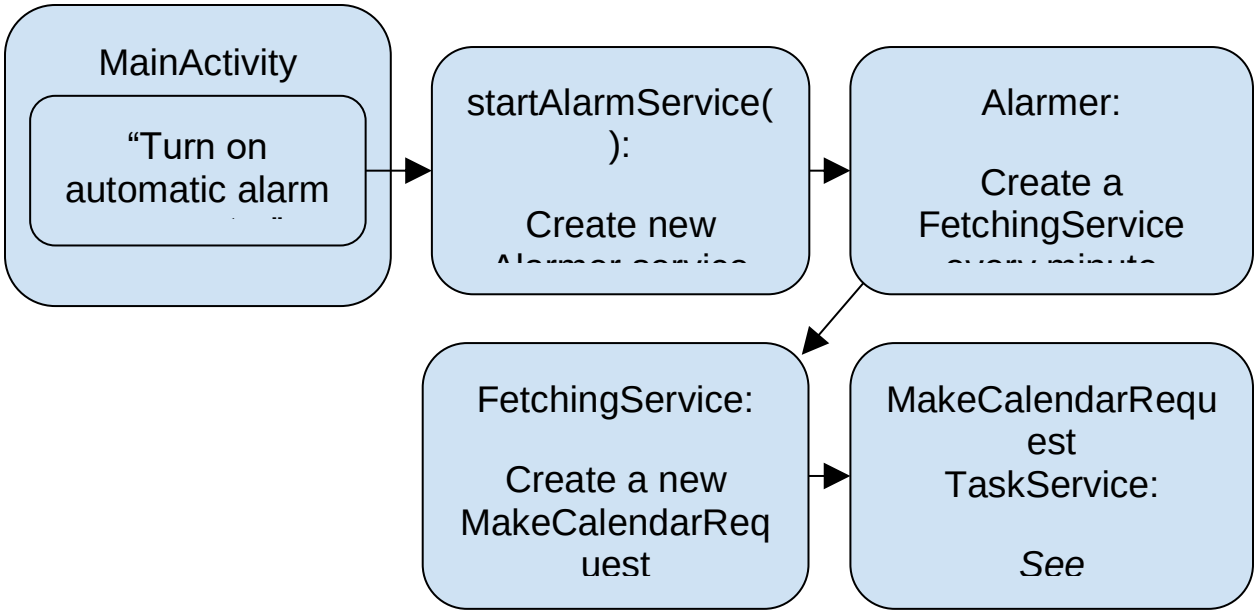


Figure 14: Automatic Alarm Generation Block Diagram

Figure 15: Picture of phone screen

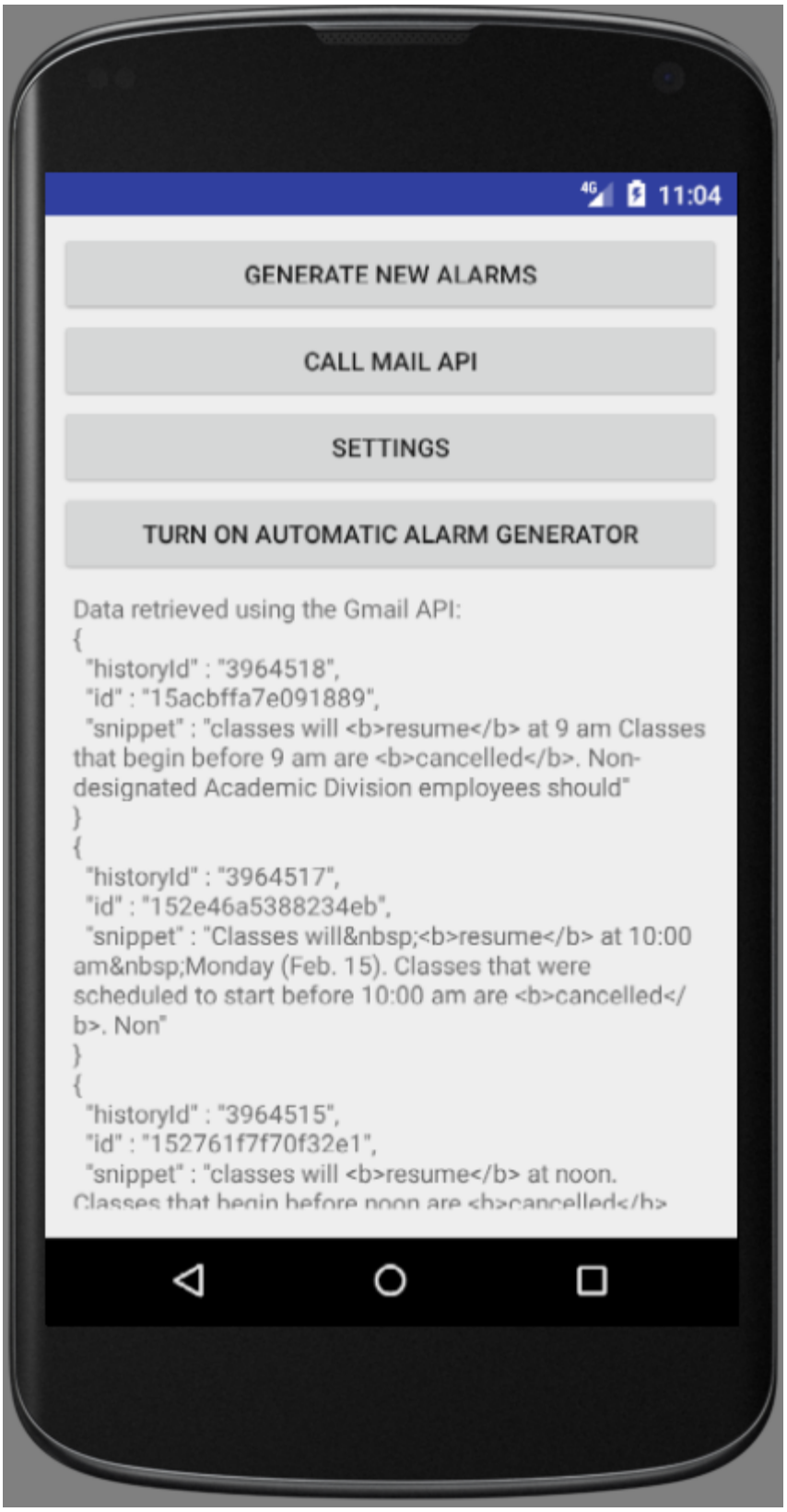


Figure 16: Email output

Figure 17: Generated Alarms

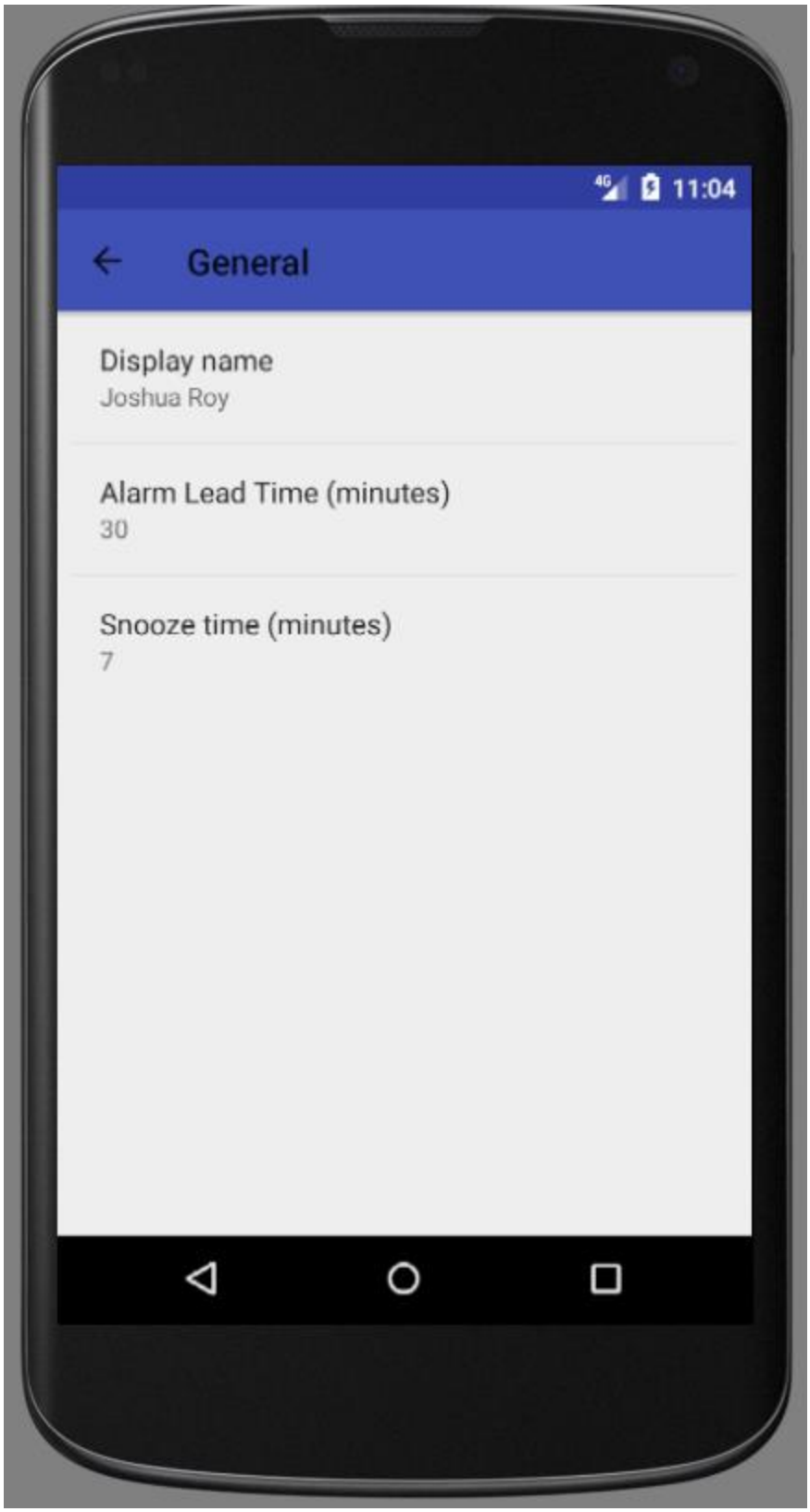


Figure 18: Settings Menu

MSP430 Software

The software for the MSP430FR5962 microcontroller was programmed using the Code Composer Studio software. The code served to connect via SPI to the Bluetooth modules and LCD display, handle all alarm functionality, and turn on the buzzer when an alarm goes off. The SPI code was taken from the software SPI code developed in Introduction to Embedded Systems. The code was modified to fit the pinout for the variation of the MSP430 used in this project, and two copies of the SPI were made for connecting both to the LCD display and the Bluetooth module that interacts with the phone application.

The Bluetooth module that interacts with the bed sensor only receives a value of 1 or 0, which means that SPI was not needed and code was written to determine if a 1 or 0 value was given at the appropriate pin on the MSP430. Based on that value, the alarm was either turned off completely or turned back on. The piezo buzzer was connected to a pin on the microcontroller and turned on by sending a square wave signal to the NPN BJT connected to the buzzer. For pin connection setup, a pin_io_setup.h file was created to make the pins easier to access and understand in the main.c code.

Upon initializing the main() function, the program initializes all pins used for the program as well as all SPI functionality, then waits for one minute before calling the phone_interact() function. This delay is to allow the display to start up and for all previously initialized alarms to be turned on if needed. The phone_interact() function uses SPI to receive the alarm information from the phone application. It turns every character sent into a byte string and then sends the byte string to the parseData() function. The parseData() function takes the byte string and breaks it down into several different arrays holding the alarm time, the snooze time for the alarm, and the user and alarm names. The alarm is then put into an Alarms structure to hold it, and there exists an array of Alarm structures to hold all of the alarms.

The alarmTimers() function is in charge of turning the alarms on based on calculations from the current time to the alarm time, and the bedSensorInteract() function determines if there is a 1 or 0 value coming into the bed sensor Bluetooth pin and, if so, waits a minute to check again and turns the alarm on if someone is still in bed. The bedSensorInteract() function is called in the alarmSound() function, which makes the sound for the alarm. While the alarmTimers() says when to turn the alarm on, alarmSound() actually sends the square wave to the buzzer. The functions timeDelay() and configureClockModule() were also included for delay handling and clock frequency calculations respectively. A breakdown of the alarm functionality is shown in the figure below.

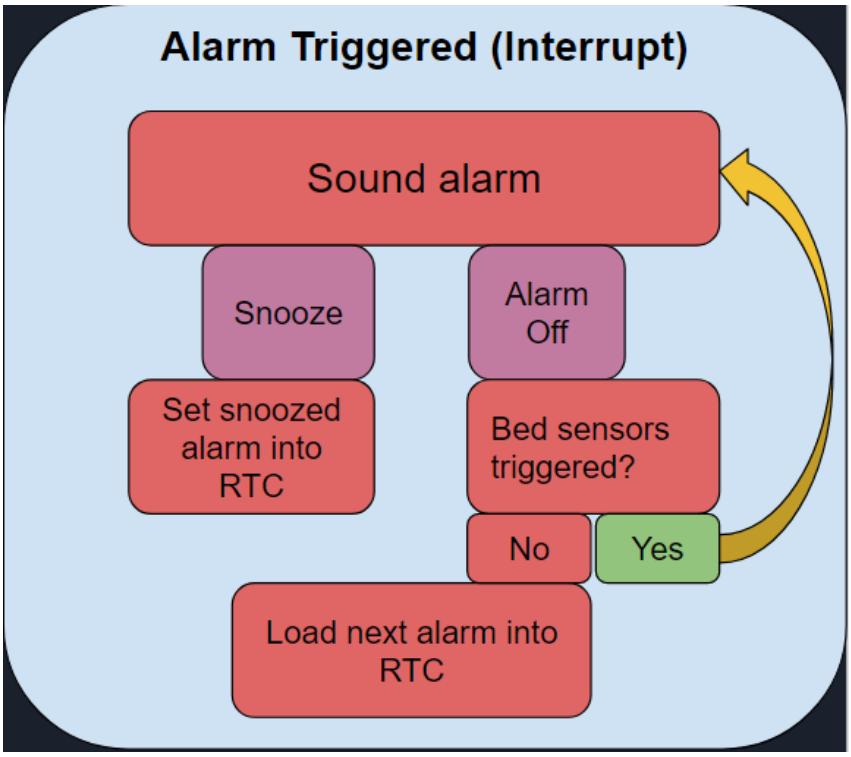


Figure 19: Flow chart for alarm functionality

RedBearLab BLE Nano Software

The software for the three RedBearLab BLE Nano v2 boards served the purpose of converting between Bluetooth and digital I/O. The first piece of software to be finished was the code for the board that receives the status of the bed sensors over Bluetooth and puts it out over a digital pin (Clock_BedSensor_RedBear.ino). It is mostly based off of the "BLE_CentralTest" example sketch provided by RedBearLab; really, the only edits were to have the system look for a BLE peripheral called "IDK_Bed_Sensors" and to modify the output to being over digital I/O instead of over the Serial UART line. If the system receives a byte with any value other than 0x00, it pulls the output pin HIGH, otherwise it pulls the pin LOW (*see Figure 20: BLE to Clock Block Diagram*). Originally, the output was to be over SPI, wherein the clock would poll the RedBearLab board for the state of the bed sensors, but it was determined in design that it would be simpler to have the RedBearLab board simply output a HIGH or LOW voltage without stimulus.

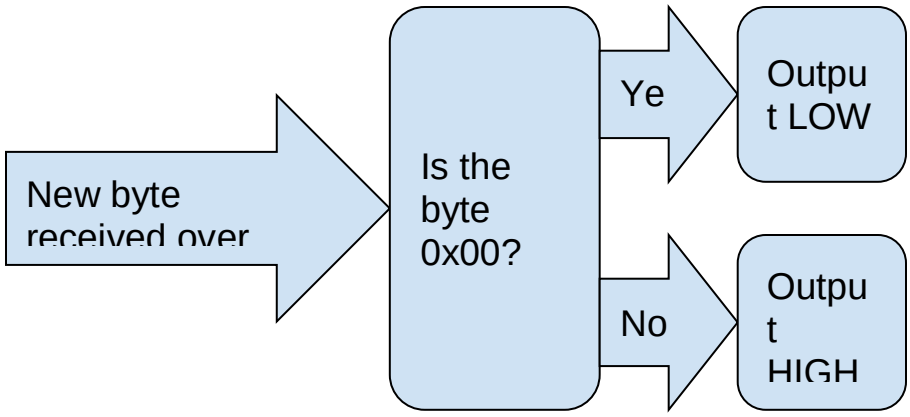


Figure 20: BLE to Clock Block Diagram

The next piece of software to be completed was the code for the RedBearLab board receiving the output of the bed sensors and putting that information out over Bluetooth (BedSensorRedBear.ino). The code was based off of the RedBearLab example sketch "BLE_HRM", an implementation of a basic counter using the Heart Rate Monitor BLE service. The primary edits made were in what data was sent over Bluetooth: Instead of simply incrementing and sending a counter, the software polls the four force sensor inputs, determines if any is above the threshold, and, if so, put out 0xFF, otherwise put out 0x00 (*see Figure 21: Bed Sensors to BLE Block Diagram*). Similar to the RedBearLab board that receives this data, this software was originally to be an SPI slave as well. However, when the capabilities of the RedBearLab BLE Nano v2 boards were understood, it was determined that the bed sensors didn't require the use of an MSP at all - instead, this RedBearLab board could handle reading the sensors through its ADC instead of the MSP reading the sensors and passing the values along via SPI.

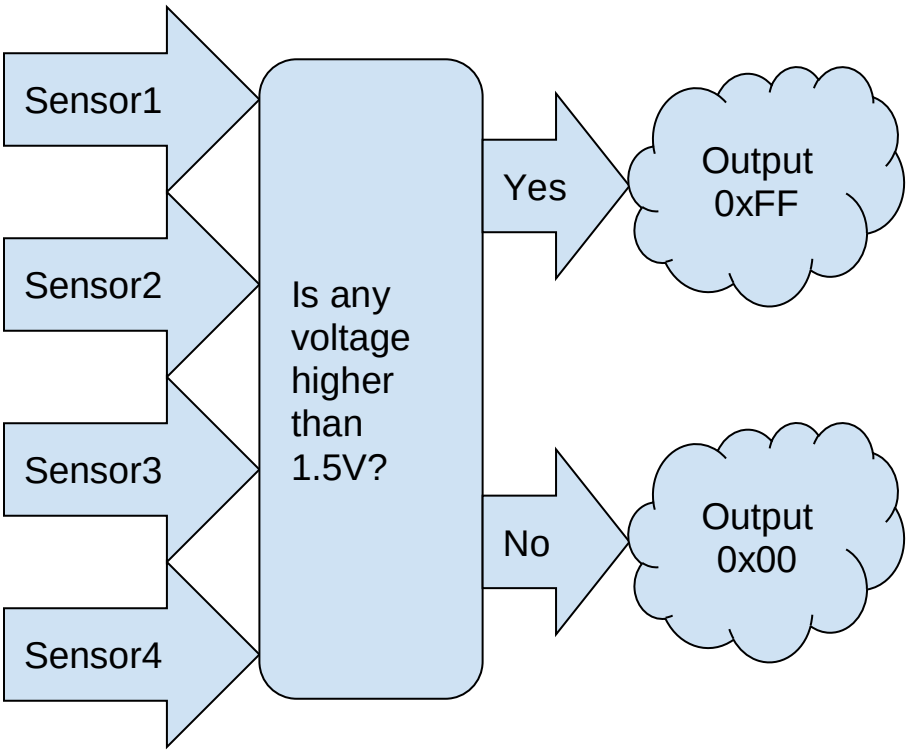


Figure 21: Bed Sensors to BLE Block Diagram

The final, and most complicated, piece of RedBearLab software was the code for communication between the Configuration Phone Application and the alarm clock ("Clock_Phone_Redbear.ino"). Unlike the software for the other two RedBearLab boards, this piece of software had to include SPI communication for sending the alarm times to the alarm clock's MSP. While the SPI slave code was originally developed as part of the software's main loop, it had to be reimplemented in an interrupt to allow the Bluetooth reception to work correctly. That Bluetooth communication was simplified by RedBearLab's example code for a peripheral that could receive messages from its master ("BLE_SimplePeripheral"). Instead of putting the received message out over the Serial UART communication line, the code was adapted to compile the received bytes into a string (ending with a terminal character 0x00). That completed string is what is then read by the sendSPIData() function, which puts each byte of the string out over the SPI lines (*see Figure 22: BLE/SPI Block Diagram*) .

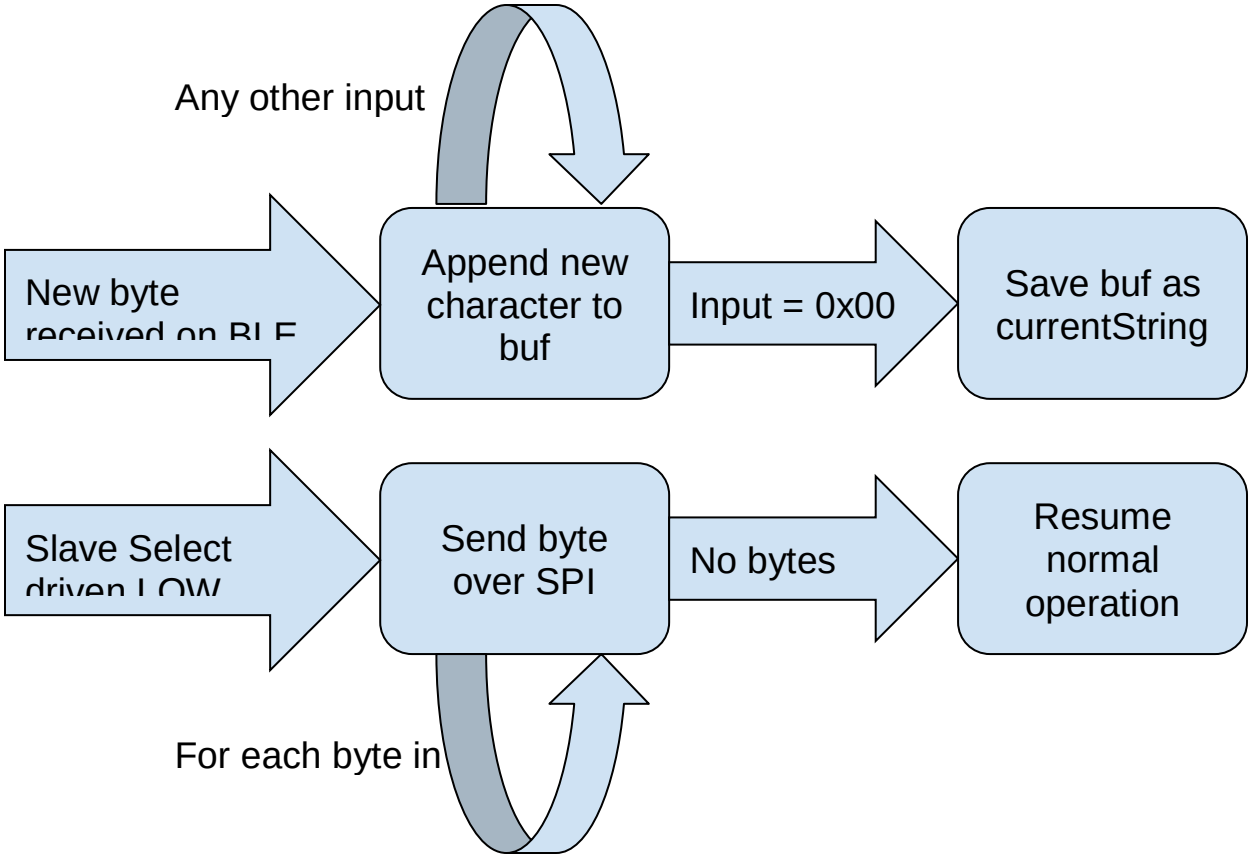
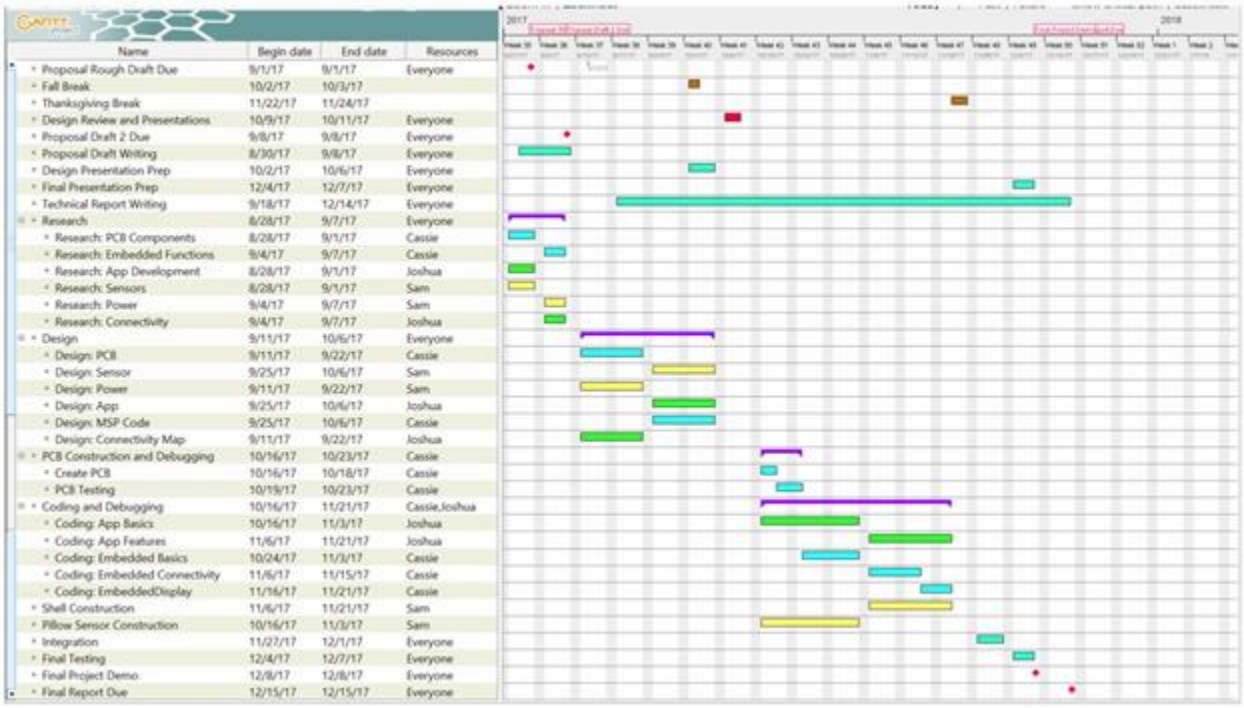


Figure 22: BLE/SPI Block Diagram

The majority of the design constraints of the RedBearLab board software were due to the requirements of the software-based SPI slave code that was built for this project. Software SPI was chosen over hardware SPI because there was no example or documentation of how to implement the chip's hardware SPI through the RedBearLab API, and software SPI is not difficult to program. However, it was found that the SPI slave code required a delay to ensure that the interrupt had time to start before the message, and if the message was sent too quickly, the RedBearLab board would misinterpret the message.

Project Timeline



Name	Begin date	End date	Resources
Proposal Rough Draft Due	9/1/17	9/1/17	Everyone
Fall Break	10/2/17	10/3/17	
Thanksgiving Break	11/22/17	11/24/17	
Design Review and Presentations	10/9/17	10/11/17	Everyone
Proposal Draft 2 Due	9/8/17	9/8/17	Everyone
Proposal Draft Writing	8/30/17	9/8/17	Everyone
Design Presentation Prep	10/2/17	10/6/17	Everyone
Final Presentation Prep	12/4/17	12/7/17	Everyone
Technical Report Writing	9/18/17	12/14/17	Everyone
Research	8/28/17	9/7/17	Everyone
Research: PCB Components	8/28/17	9/1/17	Cassie
Research: Embedded Functions	9/4/17	9/7/17	Cassie
Research: App Development	8/28/17	9/1/17	Joshua
Research: Sensors	8/28/17	9/1/17	Sam
Research: Power	9/4/17	9/7/17	Sam
Research: Connectivity	9/4/17	9/7/17	Joshua
Design	9/11/17	10/6/17	Everyone
Design: PCB	9/11/17	9/22/17	Cassie
Design: Sensor	9/25/17	10/6/17	Sam
Design: Power	9/11/17	9/22/17	Sam
Design: App	9/25/17	10/6/17	Joshua
Design: MSP Code	9/25/17	10/6/17	Cassie
Design: Connectivity Map	9/11/17	9/22/17	Joshua
PCB Construction and Debugging	10/16/17	10/23/17	Cassie
Create PCB	10/16/17	10/18/17	Cassie
PCB Testing	10/19/17	10/23/17	Cassie
Coding and Debugging	10/16/17	11/21/17	Cassie,Joshua
Coding: App Basics	10/16/17	11/3/17	Joshua
Coding: App Features	11/6/17	11/21/17	Joshua
Coding: Embedded Basics	10/24/17	11/3/17	Cassie
Coding: Embedded Connectivity	11/6/17	11/15/17	Cassie
Coding: EmbeddedDisplay	11/16/17	11/21/17	Cassie
Shell Construction	11/6/17	11/21/17	Sam
Pillow Sensor Construction	10/16/17	11/3/17	Sam
Integration	11/27/17	12/1/17	Everyone
Final Testing	12/4/17	12/7/17	Everyone
Final Project Demo	12/8/17	12/8/17	Everyone
Final Report Due	12/15/17	12/15/17	Everyone

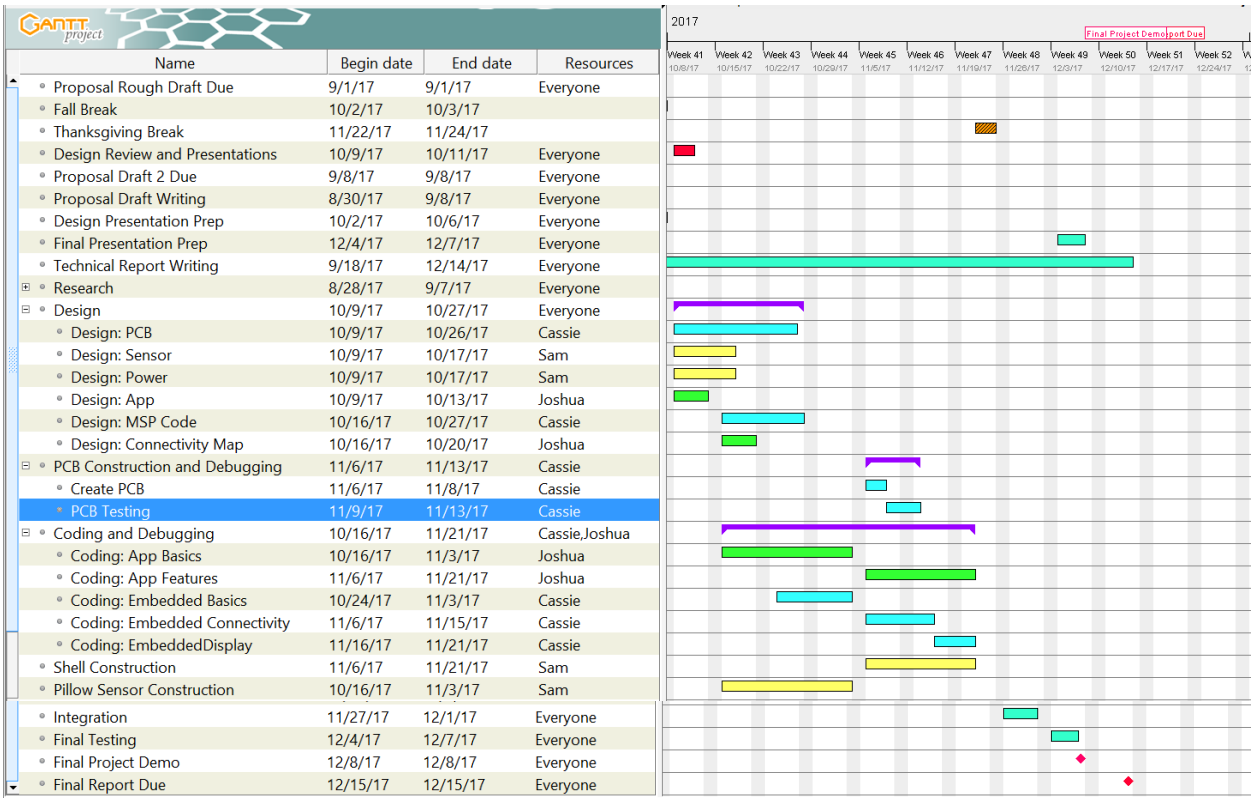
Color Key			
	Joshua Roy's Tasks		Task with Subtasks
	Holiday - No Class		Deadline
	Samuel Byers' Tasks		Everyone's Tasks
	Cassie Willis' Tasks		

Figure 23: Original Gantt Chart of Project Timeline

According to our initial timeline, practically every member was working in parallel. Sam worked on designing the bed sensors, while Joshua drafted code for the phone application, while Cassie designed the main alarm clock circuit board. Each member was assigned a list of tasks to accomplish in serial manner, though each member worked on their own sets near-independently of each other. Secondary tasks of designing a clock shell, working on bluetooth integration, and writing the microcontroller code were designated to Sam, Joshua, and Cassie respectively. These tasks were also serialized, but still in parallel to other members of the team without overlap.

As we progressed to the middle of the semester, we found our biggest adjustment to be for time. The research and early conceptualization stages had taken longer than expected, putting

us behind where we were scheduled to be. In our revised plan, future tasks were given slightly longer worktimes, while unfinished earlier parts were shuffled and reallocated to make the best use of the remaining time.



Name	Begin date	End date	Resources
Proposal Rough Draft Due	9/1/17	9/1/17	Everyone
Fall Break	10/2/17	10/3/17	
Thanksgiving Break	11/22/17	11/24/17	
Design Review and Presentations	10/9/17	10/11/17	Everyone
Proposal Draft 2 Due	9/8/17	9/8/17	Everyone
Proposal Draft Writing	8/30/17	9/8/17	Everyone
Design Presentation Prep	10/2/17	10/6/17	Everyone
Final Presentation Prep	12/4/17	12/7/17	Everyone
Technical Report Writing	9/18/17	12/14/17	Everyone
Research	8/28/17	9/7/17	Everyone
Design	10/9/17	10/27/17	Everyone
Design: PCB	10/9/17	10/26/17	Cassie
Design: Sensor	10/9/17	10/17/17	Sam
Design: Power	10/9/17	10/17/17	Sam
Design: App	10/9/17	10/13/17	Joshua
Design: MSP Code	10/16/17	10/27/17	Cassie
Design: Connectivity Map	10/16/17	10/20/17	Joshua
PCB Construction and Debugging	11/6/17	11/13/17	Cassie
Create PCB	11/6/17	11/8/17	Cassie
PCB Testing	11/9/17	11/13/17	Cassie
Coding and Debugging	10/16/17	11/21/17	Cassie,Joshua
Coding: App Basics	10/16/17	11/3/17	Joshua
Coding: App Features	11/6/17	11/21/17	Joshua
Coding: Embedded Basics	10/24/17	11/3/17	Cassie
Coding: Embedded Connectivity	11/6/17	11/15/17	Cassie
Coding: EmbeddedDisplay	11/16/17	11/21/17	Cassie
Shell Construction	11/6/17	11/21/17	Sam
Pillow Sensor Construction	10/16/17	11/3/17	Sam
Integration	11/27/17	12/1/17	Everyone
Final Testing	12/4/17	12/7/17	Everyone
Final Project Demo	12/8/17	12/8/17	Everyone
Final Report Due	12/15/17	12/15/17	Everyone

Figure 24: Modified Gantt chart for mid-semester review

What we found over the course of this project was that, while this strategy looked efficient on paper, it was poorly handled in practical use. We parallelized and divided our tasks so well that we began to work beside each other rather than with each other. This led to communication breakdowns later in the project when it came time to connect and combine the various subsystems. Had the Gantt chart been more closely integrated into the design process, better time management may have been found.

Test Plan

Phone Application Software

The Configuration Phone Application was tested feature-by-feature as development occurred, with primary debugging output on the user interface (when the system under test had access to the user interface), and all other debugging output on the debug log. It was primarily tested on a simulated Nexus 4 on Android API version 22.

Bed Sensor System

Before any components of the bed sensor system were soldered, or even breadboarded, the sensors themselves were tested in order to establish their functionality and threshold range. This was done by connecting a digital ohmmeter to the two leads on the sensor. Pressure on the sensor was then increased by incrementally layering metal discs of equal weight, allowing for the resistance to be measured at each level of pressure. (*see Figure 25: Testing sensors with increment weights*) This yielded a roughly logarithmic curve consistent with the documentation provided by the manufacturer (*see Figure 26: Sensor documentation*).

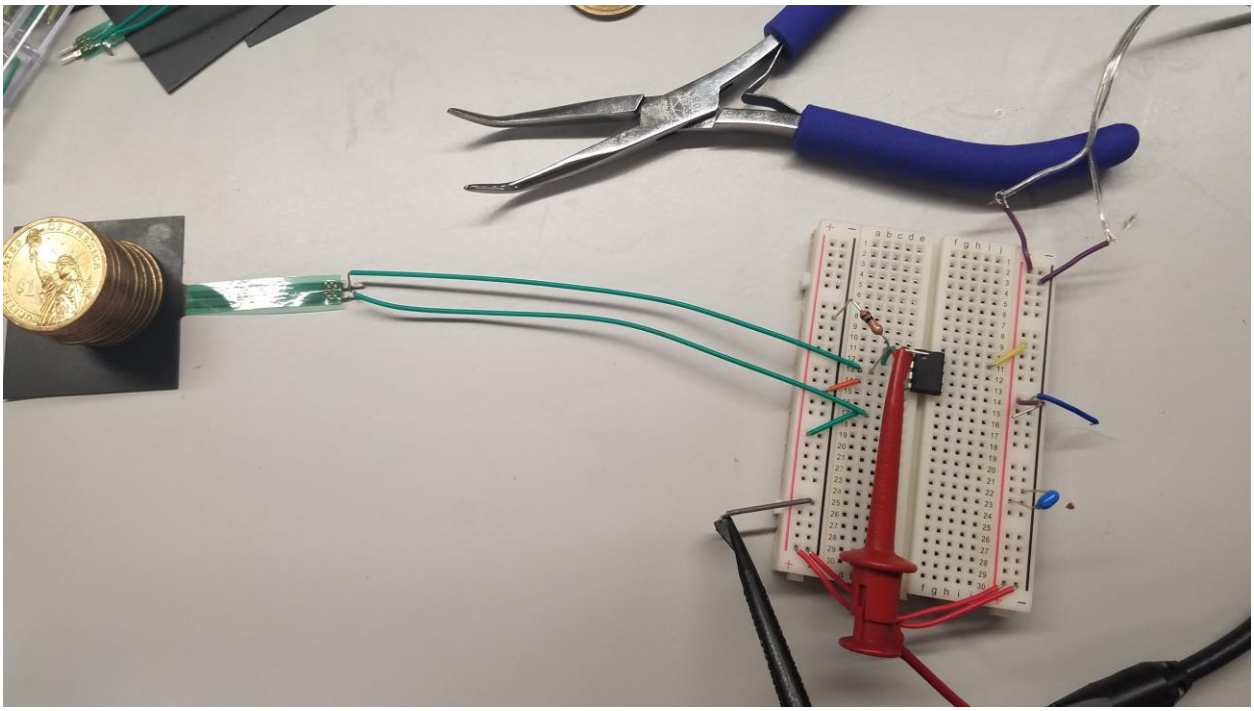


Figure 25: Testing sensors with increment weights

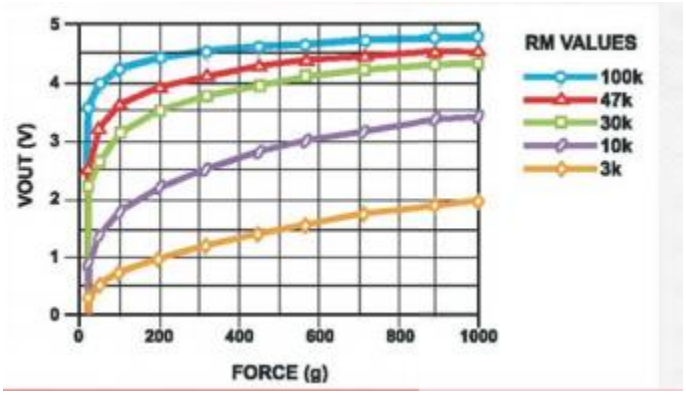


Figure 26: Sensor documentation

Next, the unity gain buffer portion of the circuit was tested. Using the Labview equipment, we passed a sine wave varying from zero to nine volts through the unity gain buffer.

Since the resistors could only decrease the voltage from the maximum input, this would simulate all possible signals which the buffer could receive. In Figure 27, the red line is the signal in, while the yellow is the signal measured at the unity gain buffer output. For reasons we had yet to determine, whenever the input signal fell below 1.3 volts, the buffer's output quickly rose to rail voltage. Though we were unable to ascertain the cause of this problem, we were able to correct it. At the time of testing, we had been using TL072ACP type op amps. When we exchanged them for LMC6482IN model op amps the problem was corrected, as shown in the Figure 28.

The Bluetooth connecting Red Bear chip was then tested. After providing it with power, a series of low and high voltages were passed into the port which would later receive the signal from the unity gain buffer. As expected, when the voltage was high, an LED activated on a second Red Bear chip, which was actively searching for the signal from the bed sensor system's chip. This indicated a successful Bluetooth communication.

Since each major component had been tested and shown to work independently, they were breadboarded together. The test from the sensor portion was repeated, now with the sensors integrated. As expected, when depressed, the LED on the receiving Red bear chip illuminated, indicating that the pressure had successfully lowered the resistance, thus raising the voltage which was likewise successfully transmitted. Varying levels of pressure were tested on all sensor pads before soldering occurred.

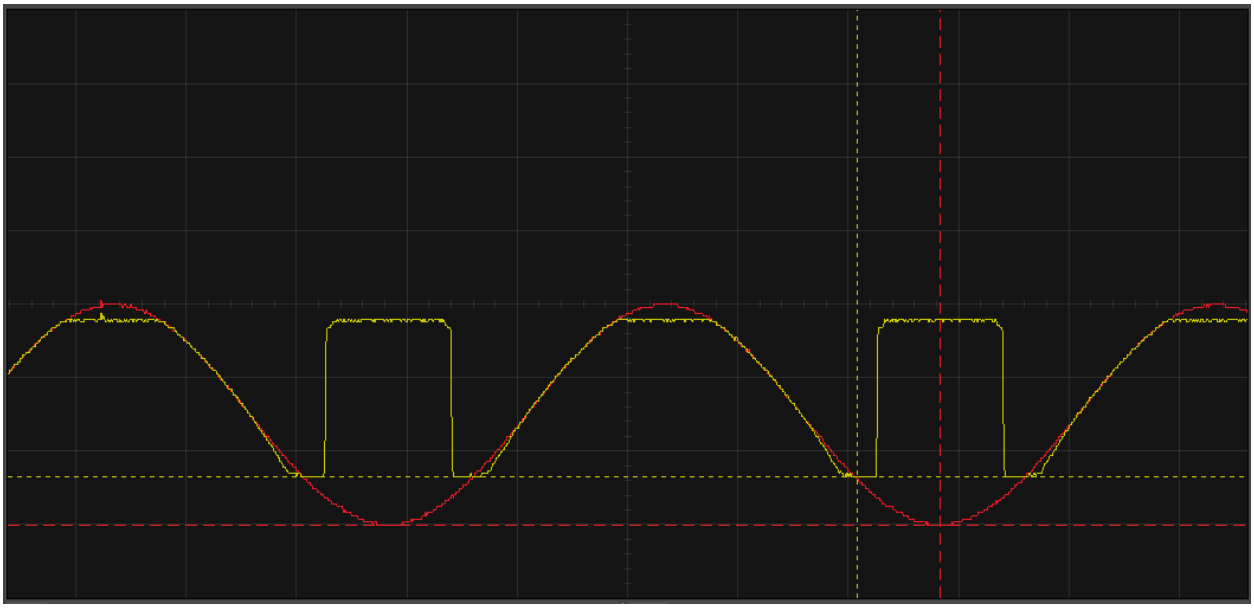


Figure 27: OpAmp Problem

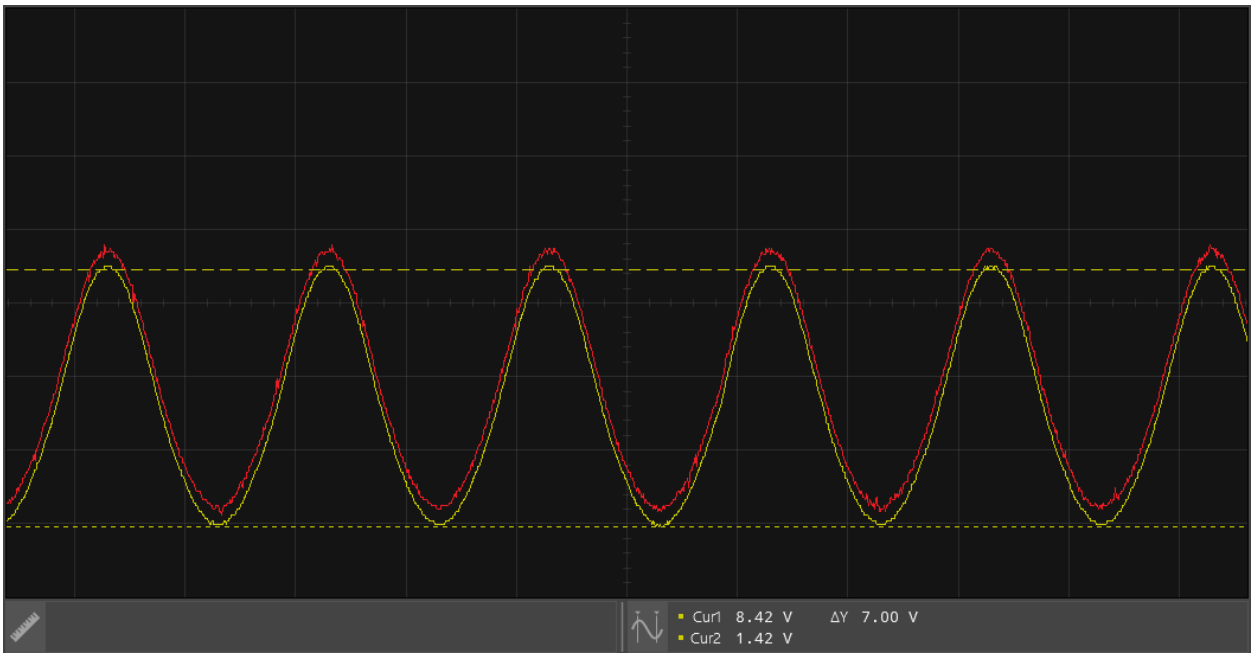


Figure 28: OpAmp Fixed

Alarm Clock

The alarm clock testing was performed as the alarm clock PCB was constructed. The MSP430 microcontroller was soldered first because it had to be taken to WWW Electronics and soldered via the pick-and-place machine. Because that component was the most volatile on the board and also required soldering first, care was taken in testing to ensure that the microcontroller was not in danger of being damaged during the testing of other components.

The designer first double checked the calculations and datasheets to ensure that all resistor and capacitor values were correct. Due to the nature of the board compromising of several designs given in the datasheets, the designer did not test the operation of these components individually, trusting that the datasheets displayed the correct values.

The designer then soldered the non-active components onto the board. This includes all resistors, capacitors, connectors, and header pins. The designer double and triple checked that each component on the board was placed correctly. The primary testing on the board came from the active components. To test these, the designer soldered them onto the board and plugged the board in to first ensure correct power. Due to a design error, the voltage regulator that ensured proper power to the board was overheated and destroyed, which meant a workaround had to be found in powering the board. A battery pack was substituted, and the designer used a multimeter to ensure proper 3.3V powering throughout the board, which was successfully determined.

The JTAG interface was tested next, with the designer using a JTAG system to attempt to load code onto the microcontroller. Possibly due to a short caused by the overheating of the voltage regulator, the JTAG interface could not successfully program the microcontroller. Debugging was performed by checking all of the connections on the board as well as the pin diagrams for the JTAG interface. Due to timing constraints, a successful solution was not found.

Finally, the designer tested the Bluetooth headers by using a multimeter to probe the headers and ensure proper voltage, which was found. The designer then plugged a Bluetooth module into the board to determine if the board was powered. The microcontroller code was also tested, and any bugs were removed so the code was clean. Being unable to program the microcontroller, however, severely limited the amount of testing that could be performed with the code.

RedBearLab BLE Nano v2 Boards

The RedBearLab boards were tested during development, primarily through writing code that was functionally similar to the final product in a form that was more testable. The first system-for-test that was developed was a series of three RedBearLab boards, with the first as an SPI Master, the second an SPI slave and Bluetooth peripheral, and the final as a BLE Central. The SPI Master would alternate every half second between sending a command to turn on or off the slave's LED. The slave would then pass that command along using the same BLE Heart Rate Monitor service that would later be used as the service for communicating the state of the bed sensors. When all three boards lit their LEDs at the same time (with some delay for the third board, which had to rely on the slower BLE protocol), it was clear that the testing system was working as desired. That testing system actually proved to be more sophisticated than the bed sensor BLE communication pair, as the bed sensor's RedBearLab board did not end up being an SPI slave.

This testing system delivered a useful system breakdown, as the point of failure clearly showed whether SPI or BLE was the failing subsystem (and if both were shown to fail, fixing the SPI implementation could reveal a correct BLE implementation). At first, the primary failures were due to SPI communication; the slave software SPI system proved to be too slow to handle the master's communications, which were easily verified using a VirtualBench to listen in on communication. One useful debugging tool was to toggle the MOSI SPI line whenever the slave checked the state of the clock line, because it occurred very quickly, unlike Serial UART debugging messages. These line toggles were also easily read by the VirtualBench and could be easily used to find the time allowance that the slave required, both between being selected and the message being sent as well as the time between clock toggles.

Integrated System

There was no testing of the entire integrated system, as the system was never finished. This was, however, the primary focus of the original testing plan, so the usefulness of that subdivision process could not be verified, but it is still expected to be a useful system breakdown.

Final Results

There were four metrics for success considered in the original project proposal for the IoT Alarm Clock. In the case of the application, and bed sensor functionality, the design met or exceeded expectations required under the guidelines of an A. For the app connectivity, alarm/clock, and display, the design unfortunately, for reasons that will be explained below, met only the expectations required of a D grade.

Grade	Bed Sensor	Alarm/Clock	Display	App/Connectivity
A	Sensor works and changes alarm	Alarm can be set and go off within 1 min of set time	Notifications are mostly correct with some lag	App connects with phone with almost always (>80%), and app can automatically adjust alarms
B	Sensor works but doesn't change alarm	Alarm can be set, clock displays, alarm goes off at wrong time or without noise	Some notifications don't display or display incorrectly, some lag	Connects often, user can edit alarms, automatic alarms can be generated from a calendar, and weather data is sent to clock
C	Sensor works on occasion, doesn't change alarm	Clock displays, alarm can't be set or goes off when not set	Notifications are incorrect or display at wrong time / with too much lag	Connects intermittently, user can create an alarm
D	Sensor does not work	Clock display is not correct, no alarm functionality	Notifications do not work, display is unhelpful	Does not connect, features mostly non-operational

Figure 29: Original success metrics

The alarm clock PCB had multiple issues that unfortunately led to the PCB being mainly unfunctional. The MSP430FR5962 microcontroller was successfully installed on the PCB, and all other components of the board were found to be functional and correctly powered. The JTAG interface that was designed to program the microcontroller, however, had a short in the wiring. This short resulted in the designers being unable to program the MSP430 microcontroller. Though the microcontroller could not be programmed, all of the code to handle the alarms was

written, and it is believed that the code would be 90% functional had the designers been able to program the microcontroller.

Another problem that arose with the alarm clock PCB was the choice of linear voltage regulators. Research was conducted to determine the number of volts supplied by a wall outlet, which was intended to power the PCB. Unfortunately, the designer accidentally read the information incorrectly and assumed 60V as the voltage output of the wall outlet. The actual voltage is 120V, and it is believed the mistake was made because 60Hz is the frequency of wall outlet voltage, and the units were read incorrectly. The linear voltage regulator chosen was rated for a maximum of 60V, so when it was plugged into the wall outlet, the voltage regulator along with a capacitor that was placed in parallel to the regulator both overheated and were destroyed. A battery pack was then acquired and used in place of the wall outlet in order to ensure proper voltage, which was achieved.

With the alarm clock, the final problem was the connection to the LCD display. The designers purchased only one ribbon cable capable of connecting the display, and the connector on the board side was the incorrect footprint and therefore too large to connect to the correct ribbon cable. This error was in part due to an incomplete datasheet for the display, which did not list the exact connector used or which ribbon cable should be purchased for connectivity. In the process of modifying the cable to fit into both connectors, the cable was destroyed. The main lesson learned in the alarm clock mistakes was extreme attention to detail in every step of the design process, from research to assembly. A secondary lesson was learned in time management, as the board had to be rush ordered, with some components being overnighted in order to receive them before the deadline. The final board can be seen in Figure 30 below.

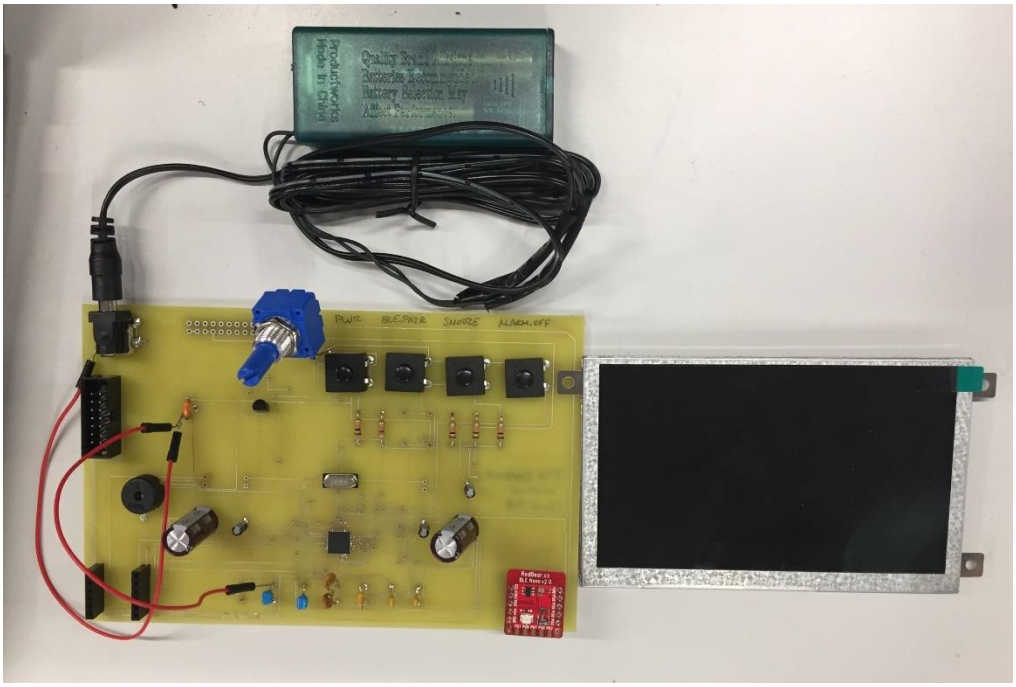


Figure 30: Final alarm clock PCB

The App was unable to connect with its associated RedBearLab board because there was not enough time to actually implement the Bluetooth communication. All of the information processing around the Bluetooth communication was completed: the string to be sent was prepared as was the system to receive the string on the RedBearLab chip. Unlike the previous failures, this was not the case of faulty product or design, but simply not devoting the proper amount of time to the development of the system.

The Bluetooth chips were able to successfully communicate with the bed sensor board and were able to successfully receive a byte string from the phone application. This was shown in the demo as a light turned on whenever the bed sensors were triggered and a computer was connected to the phone Bluetooth chip to show the byte string received by the chip from the phone.

The bed sensor board was also able to function in its entirety. The sensors correctly read whether there was a person on the bed via voltage changes. This data was then sent to the Bluetooth chip on the bed sensor board which was able to correctly send either a 1 or 0 value to the Bluetooth chip on the alarm clock board to determine if there was a person in bed.

The most important takeaway from this experience would be the importance of communication in the early stages of design. Better communication early on would have significantly reduced the number of problems we later encountered. Had we made more of our tasks cooperative rather than specialized, I am sure that we would have achieved our goals in better time and to a higher degree of personal satisfaction.

Costs

With everything taken into account, the total cost to manufacture a single instance of this project was $932.13. While all parts of the system contributed to the overall cost in small amounts, the overwhelming majority of the cost could be attributed to several key parts; namely, the printed circuit boards, the Riverdi display, and the RedbearLab chips. Together, these composed over 80% of the project's cost. However, the primary contributor to this (i.e. the printed circuit boards) drops drastically in price when ordered in bulk.

To simulate this, we calculated the expected cost of production if all parts were bought in bulk orders of 10,000, which can be found in the Appendix as Chart A. When ordering in bulk, the price per printed board dropped from over $325 per board to $1.36 (for the bed sensor board) and from over $370 to $0.96 (for the alarm clock board). While other components also had reduced bulk prices, this was the most substantial difference. According to our calculations, producing 10,000 instances of this project would have a resounding cost of just under $1.79 million, which comes to $179.02 per board, almost 80% less than the original price.

Even with these conditions, the biggest cost is still the Riverdi display, which remains at a fixed price of over $70 regardless of quantity ordered. Potentially a smaller and thus more inexpensive screen could be sourced, but that is beyond the scope of this assessment.

Another cost incurred, but not calculated here, would be labor and other costs related strictly to the manufacturing process. While these would occur additional expenses, most of the work is simple soldering, which can be easily automated by machines.

Future Work

The clearest issues which we ran into were time management and communication. Though our schedule was thoroughly planned out from the start, we did not give ourselves enough buffer time to adjust the timing of our tasks. Everything took longer than expected, especially research and designing, leaving the physical tasks of building and integration crammed into the final weeks. This also left us in a situation where sacrifices and hard decisions had to be had to be made, when the looming deadline made it difficult to order new or replacement parts and boards. When we encountered problems (either from a cable having a short, or a board's pins being arranged upside down) rather than order new correct parts, our limited time forced us to make do and create workarounds as best we could.

If this project were to be expanded upon, there are several directions in which it could potentially go. One avenue of advancement would be enabling the phone application to read the user's emails. This was an aspect of the original idea which we abandoned early on as we gained a better understanding of the difficulty involved. Although such a process could be done, it was more time consuming than we could budget for given our timeframe. Such an addition would allow the application to amend alarm times based on the content of incoming emails, such as if a class is cancelled or a meeting delayed.

Further advancement aside, there are also pitfalls which we encountered which pursuers of this project should take heed of. One such is the delicate sensitivity of the crystal chip. When designing the board layout, the crystal should be placed as close to the parts it interacts with as possible. As we found out the hard way, its signal is easily swayed by the currents in the traces running near to it.

References

[1] Willis, C., Byers, S. and Roy, J. (2017). *IDK Development Solutions IoT Alarm Clock*. Charlottesville, VA, pp.1-2.

[2] L. Columbus, "Roundup Of Internet Of Things Forecasts And Market Estimates, 2016," *Forbes*. [Online]. Available: https://www.forbes.com/sites/louiscolumbus/2016/11/27/roundup-of-internet-of-things-forecasts-and-market-estimates-2016/. [Accessed: 31-Aug-2017].

[3] "Magic Mirror: Part I - The Idea & The Mirror," *Xonay Labs | Michael Teeuw*. [Online]. Available: http://michaelteeuw.nl/post/80391333672/magic-mirror-part-i-the-idea-the-mirror. [Accessed: 31-Aug-2017].

[4] *WITTI - BEDDI Style | App Enabled Smart Alarm Clock with Bluetooth Speaker & USB*

Charging Station.

[5] *Vobot LED Dispaly Smart Radio Alarm Clock Speaker with Amazon Alexa, Wireless Portable Audio Player with Sleep Sound and Battery*.

[6] "Bonjour | Smart Alarm Clock with Artificial Intelligence," *Kickstarter*. [Online]. Available: https://www.kickstarter.com/projects/1450781303/bonjour-smart-alarm-clock-with-artificial-intellig. [Accessed: 31-Aug-2017].

[7] "802.11-2016 - IEEE Standard for Information technology--Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," Institute of Electrical and Electronics Engineers, Dec. 2016.

[8] "Hypertext Transfer Protocol -- HTTP/1.1," Internet Engineering Task Force, Standard, Jun. 2014.

[9] "1149.1-2013 - IEEE Standard for Test Access Port and Boundary-Scan Architecture," Institute of Electrical and Electronics Engineers, May 2013.

[10] "Universal serial bus interfaces for data and power – Part 1-3: Universal Serial Bus interfaces – Common components – USB Type-CTM cable and connector specification," International Electrotechnical Commission, Aug. 2016.

[11] "Specifications," *Bluetooth*. [Online]. Available: https://www.bluetooth.com/specifications. [Accessed: 15-Dec-2017].

[12] J. Perry, "IPC-7351C Revision Goals," Association Connecting Electronics Industries, Oct. 2012.

[13] D. Landes, *Revolution In Time*, 2nd ed. Harvard University Press, 2000.

[14] J. Chipchase, "Alarm Clock," 7304563, 04-Dec-2007.

[15] C. Yang, S. Kim, G. Choi, and E. Lee, "Method and apparatus for setting snooze interval in mobile device," 9703269, 11-Jul-2017.

[16] Y. Zhu, "Setting an alarm clock on a smart device," 9684284, 20-Jun-2017.

[17] C. Robertson, "Wireless wake-up alarm with occupant-sensing apparatus," 9679462, 13-Jun-2017.

Appendix

Chart A - Expected Production Costs

Part Number	Part Description	Price per 1	Price per 1/10000	Price for 10000
Bed Sensor Parts				
LMC6482IN/NOPB-ND	LMC6482IN Op Amp	$1.73	$0.85	$8,500.00
LMC6482IN/NOPB-ND	LMC6482IN Op Amp	$1.73	$0.85	$8,500.00
AE9986-ND	Op Amp Socket Connector	$0.19	$0.08	$766.60
AE9986-ND	Op Amp Socket Connector	$0.19	$0.08	$766.60
445-174875-1-ND	0.1µF Capacitor	$1.21	$0.36	$3,614.00
445-174875-1-ND	0.1µF Capacitor	$1.21	$0.36	$3,614.00
CF14JT10K0CT-ND	10kΩ Resistor	$0.10	$0.00	$41.25
CF14JT10K0CT-ND	10kΩ Resistor	$0.10	$0.00	$41.25
CF14JT10K0CT-ND	10kΩ Resistor	$0.10	$0.00	$41.25
CF14JT10K0CT-ND	10kΩ Resistor	$0.10	$0.00	$41.25
952-2262-ND	1x2 Header Pins	$0.11	$0.37	$367.50
1597-1497-ND	Redbear BLE Nano V2	$17.85	$17.85	$178,500.00
485-1075	Pressure Sensor	$7.95	$7.95	$79,500.00
485-1076	Pressure Sensor	$7.95	$7.95	$79,500.00
485-1077	Pressure Sensor	$7.95	$7.95	$79,500.00
485-1078	Pressure Sensor	$7.95	$7.95	$79,500.00
	Circuit Board Printing	$326.88	$1.36	$13,553.03

Alarm Clock Parts				
MSP430FR596x	MSP Microcontroller	$6.43	$3.41	$34,073.00
42-1018-ND	Power cord	$4.48	$2.71	$27,135.00
MIKROE-2169	Riverdi Display	$71.40	$71.40	$714,000.00
1597-1497-ND	Redbear BLE Nano V2	$17.85	$17.85	$178,500.00
1597-1497-ND	Redbear BLE Nano V2	$17.85	$17.85	$178,500.00
CW181-ND	Pushbutton	$0.74	$0.45	$4,455.00
CW181-ND	Pushbutton	$0.74	$0.45	$4,455.00
CW181-ND	Pushbutton	$0.74	$0.45	$4,455.00
CW181-ND	Pushbutton	$0.74	$0.45	$4,455.00
1597-1498-ND	BLE dongle	$31.88	$31.88	$31.88
490-7556-1-ND	4.7uF capacitor	$1.22	$0.38	$3,757.00
490-7556-1-ND	4.7uF capacitor	$1.22	$0.38	$3,757.00
445-2525-1-ND	piezobuzzer	$0.72	$0.27	$2,727.00
493-4771-1-ND	10uF electrolytic capacitor	$0.78	$0.20	$1,991.86
493-4771-1-ND	10uF electrolytic capacitor	$0.78	$0.20	$1,991.86
91A1A-B28-B15L-ND	Potentiometer	$5.91	$2.96	$29,550.00
535-10207-1-ND	Crystal	$0.36	$0.15	$1,540.00
493-10419-1-ND	1uF electrolytic capacitor	$0.28	$0.04	$420.95
493-10419-1-ND	1uF electrolytic capacitor	$0.28	$0.04	$420.95
493-10419-1-ND	1uF electrolytic capacitor	$0.28	$0.04	$420.95
399-4521-1-ND	1uF capacitor	$0.41	$0.08	$833.00
BC1101CT-ND	100nF capacitor	$0.19	$0.02	$241.10
BC1101CT-ND	100nF capacitor	$0.19	$0.02	$241.10
CF14JT47K0CT-ND	47k resistor	$0.10	$0.004	$41.25
CF14JT10K0CT-ND	10k resistor	$0.10	$0.004	$41.25
CF14JT10K0CT-ND	10k resistor	$0.10	$0.004	$41.25
CF14JT10K0CT-ND	10k resistor	$0.10	$0.004	$41.25
CF14JT10K0CT-ND	10k resistor	$0.10	$0.004	$41.25
CF14JT10K0CT-ND	10k resistor	$0.10	$0.004	$41.25
CP-002BHPJCT-ND	Power connector	$1.44	$0.60	$6,011.60
CP-002BHPJCT-ND	Power connector	$1.44	$0.60	$6,011.60
	Circuit Board Printing	$373.55	$0.96	$9,599.88
			Estimated	Actual
	Total:	$925.70	$207.82	$1,756,128.71
				$175.61