A Novel Wireless Solution for Acquiring and Representing Data on Current and Past Parking Trends.

Victor Avila, Austin Hinton, Derek Moore, Ian Noy, Jason Parkin
Dept. of Electrical and Computer Engineering, School of Computing, University of Utah
vic.avi88@gmail.com, a33hinton@gmail.com, djmoore89@hotmail.com,
ianj.noy@gmail.com, jasonparkin36@gmail.com

***Abstract*— Parking spaces can be hard to find in large lots. Large amounts of time can be spent trying to find an available space, and when one is found there is a risk of losing the space to another person. We propose a possible solution to finding available spaces: an embedded circuit which uses IR sensors to determine if a spot is taken or available. This information would then be transmitted to a web server which would then provide a user interface showing available spots. The results of the development of the system are presented and analyzed, as well as a discussion of the system as a whole. We believe that this system will mitigate the time wasted in finding a suitable parking space.**

I. Introduction and Motivation

A. Introduction

As the progression of time marches forward, the population of our world continues to increase. As such, there is a consistent increase in the number of drivers on the road every year. From 2008 to 2013, the total number of licensed drivers in the state of Utah increased from 1.75M to 1.9M [1] [2]. The increase in drivers using public roadways has put additional stress to an already stressed parking framework in place in our state. This stress has been notably perceived by anyone who commutes to the University of Utah (The U), as there has been an increase in the number of student and faculty commuters who use private transportation as their main means of travel to and from said institution. In the same time frame stated above, the number of students registered at The U increased from 28,211 in 2008 to 31,515 in 2014, an increase of 11% [3] [4]. Of these 31,515 students, 23,907 were undergraduate, and 87% of undergraduate students identified as commuters, i.e., not living on school-owned facilities [5].

The staggering number of individuals commuting to the The U means that finding a suitable parking location can be a major inconvenience. There is a considerable amount of time that must be spent in search of a parking space for anyone not arriving at The U before sunrise. This time wasted can lead to missed examinations and being late to potentially important meetings. The problem is such that some commuters have conceded defeat and either begin their travel to The U hours before they need to do so, or have opted to take advantage of public transportation. However, use of these alternatives has not mitigated the parking problem faced by those who elect to drive to The U. Furthermore, there are circumstances in which these alternatives are not an option, such as last-minute meetings or, more realistically, being late.

In response to this issue, our group aims to mitigate the headaches that are associated with parking at The U, primarily with respect to finding a suitable parking location. We aim to engineer a parking space monitoring system that tracks whether a parking space is occupied or vacant. The system will be comprised of sensory nodes that will communicate with one-another, as well as with a master node. This master node will transmit all the data for an entire parking lot to an off-site server, where it will be processed and the status of each parking space will be displayed graphically via a user-web interface. Users of the system will be able to access the web service before they depart on their commute and formulate a parking strategy, thereby mitigating the time lost in search of an empty parking space.

While there are other parking solutions implemented today, none have the resolution that we are aiming for. Many of these systems that are in use today are essentially counters that keep a running count of the total number of cars that pass through an entry point. While these systems seem adequate, they are missing several key pieces of data:

1) Where are the taken spaces concentrated?
2) What is the total number of parking spaces?
3) Cars that have vacated spaces, but not yet exited the lot/structure.
4) Historical analysis of data for a specific time frame.

For these reasons, we believe that our system is an improvement upon existing systems that implement similar functionality.

The system will be demonstrated in real-time during the Senior Demo Day in December. We will have the fully-functional web service displaying data from the server, which will be acquiring data from a live unit monitoring 4 spaces. The unit will be located in the parking lot of the Merrill Engineering Building (MEB). If for some reason we are unable to place the system in the parking lot, we will instead place the system in the demo room, or at a hallway intersection to simulate objects being detected by the system.

B. Motivation

The motivation behind this project comes from frustration, specifically the frustration that is associated with finding a parking space in the middle of the day. Each member of the group has had to, at one point or another, arrive at The U in the later hours of the morning or midday. During these times, it is near impossible to find a decent parking spot with a quick scan of the desired lot. The result is that we, among others looking for parking, must drive up and down the rows of cars in hopes of finding a spot that has been recently vacated or has been overlooked. The act of crawling through the lot not only wastes time, it also contributes to the release of excess pollutants into the atmosphere. We hope that this system will mitigate some of these effects by informing individuals of open spaces, thereby reducing the searching, which will reduce the production of exhaust gases that are a result of crawling around the parking lot.

Additionally, we believe that this idea has the potential to be marketable. Our primary target market would be institutions of higher learning with a high percentage of commuting students. Our secondary target market would be amusement park operators. Both of these entities have tremendous problems with parking, specifically with the difficulties associated with finding a suitable location.

II. Background

Parking monitor systems have been around for a while. The systems come in a variety of technologies that are designed to monitor parking lots. Most of the systems use image processing to identify available spots. Other systems count the number of cars that have entered the lot and keep a running total of free spaces. The parking lot monitoring system that this paper describes is derived from a combination of these already existing systems.

A. Spot Identification

Patent US7893847 claims that with image processing and a parking availability determiner, their system can identify open spots [6]. The system uses image processing to identify parking spaces with the use of symbols and relays the information over a network to a central system. Other systems use image processing to estimate the total number of open spots available.

Another type of system monitors the flow of traffic entering and leaving the parking structure and keeps a count of spot availability. This type of system can be seen at the City Creek Shopping Center in Salt Lake City, Utah. The system keeps a running total of open spots per level of the parking structure and displays the numbers on monitors. This type of system is popular for larger parking structures.

B. Spot Reservation

Patent US20140249742 A1 describes a way in which a driver gets matched to an available parking spot. The software is designed to estimate the time of arrival of the driver and reserves the space [7]. Other types of spot reservation is available through web browser assistance. The system offered by www.theparkingspot.com allows for users to reserve a parking spot at select airports.

III. Proposed Work

A. Baseline Deliverables

The system will be comprised of two custom printed circuit boards. One board design will be designated as a master node, and the other will be designated as a slave node. The slave node will be responsible for the acquisition of data for 4 parking spaces in total. The monitoring of these spaces will be done with the use of infrared (IR) sensors connected to the PCB. The slave boards are limited to monitoring 4 spaces per slave because of the fact that parking spaces are easily represented in quadrants, and because of hardware constraints to be discussed later in this proposal. The slave nodes will communicate with the master node by use of the IEEE 802.15.4 communications protocol, known as ZigBee and discussed later in this section.

The master nodes will be responsible for accumulating the data from the slave nodes and transmitting the data to the database. The transmission of the collected data will happen over Wi-fi, thereby affording us the use of a widely used communications protocol. The master node will be able to track at most 32 individual units, each of which will be end-point slave devices.

The master and slave PCBs will each have headers so that an XBee−PRO® 900HP radio can be connected directly. The radios will be used to implement a master−slave communications network via the Zigbee protocol. This communications protocol was selected due to its low power consumption and mesh network capabilities. A rechargeable battery will be used to supply power for both master and slave devices. The master node will have the CC3200 component provided by Texas Instruments to communicate via Wi-Fi. Fig. 1 shows a high-level diagram of the communication between slave and master nodes, and the master node to the server. Additionally, fabrication of a plexiglass case is being considered to ensure the hardware is protected from environmental exposure.

At the present time, a server has been setup to host the data being transmitted by the master nodes. The data will be processed and stored in a database to await usage. The data will be used by the end user via a web interface. Currently, the website is being constructed to display the state of the parking spaces on a graphical representation of the parking lot. Additionally, the website will guide the user from the entrance of a given parking lot to the closest available spot.

For the demonstration of the system, we will have a single master node monitoring two slave nodes, for a total of eight parking spaces monitored. The reason for the limited scale of the project is purely monetary. The system should be able to scale up to an almost arbitrary number of slave and master nodes, and the ZigBee protocol has been proven to scale up to 400 nodes [8].

B. Stretch Goals

It is a goal of ours to develop this project in a way that the end product is marketable. However, the primary goal is to showcase our engineering ability in designing a system

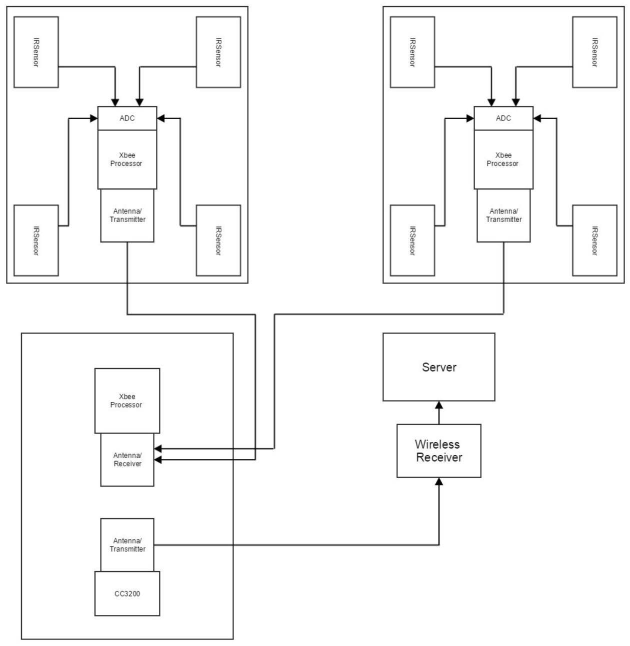


Fig. 1: High-level block diagram of baseline deliverables.

that has the potential to become a marketable product. If time permits, the following optimizations to the system will be implemented, thereby requiring more engineering, as well as potentially increasing the systems marketability.

The first goal is to create a self-sustaining product by powering the entire solution with energy harvested from solar panels. These panels would also be used to charge the battery packs, and the battery would be converted to a backup power supply. This enhancement would potentially require a rework of the currently planned power system. Additionally, we desire to develop a user-friendly mobile application for the Android operating system. This application would be a companion to the web interface, and each will display the same data. Finally, we want to provide extra features to our end application. One extra feature has been discussed is the ability to reserve parking spaces ahead of time for a nominal fee. This feature would provide useful for commuters that live far from their destination and would generate extra revenue for the owner of the parking lot.

IV. Schedule and Work assignments

A. Schedule

A rough schedule follows:

1) By the end of May, we intend to complete the PCB design and schematic, the design will be sent for fabrication, and parts will be ordered.
2) By the end of June, receive first iteration of PCB and assemble it.
3) By the end of July, the transmitting station, and server/receiver will be assembled. We will also be testing and verifying our board.
4) By the end of August, the web application will be started with continued testing on the boards and receiver as needed.
5) In September, we will continue testing.
6) In October, we will start the formal report, and continue testing as needed.
7) By November, testing is expected to be done with a finished product. The remaining time will be spent on finishing the formal report.

B. Work Assignments

To better facilitate the development of the project, the project has been divided into three major components; embedded software development, PCB design and power system design, and the server back−end and web interface. Each member of the team has been assigned to one of these specific tasks to act as a product owner and project lead in their corresponding assignment. The assignment decisions were based on taking into consideration the talents and engineering strengths of each individual, as well as personal interest.

Product ownership of the embedded software development is assigned to Victor Avila and Austin Hinton. These two individuals have expressed an interest in embedded system design, as well as wireless communications. Specifically, Austin will be primarily responsible for any and all ZigBee protocol related software, and Victor will be responsible for the Wi-Fi aspects of the project. Jason Parkin expressed an interest in analog digital circuit design, and is therefore designated as the primary point of contact for the power system portion of the project. Derek Moore expressed interest with PCB design, and has proven to be proficient with the Altium Designer tool, therefore he will be designated as the point of contact for the PCB design and the schematic design for the slave and master nodes PCBs. Finally, due to his workplace experience, and expressed interest, Ian Noy will be responsible for the server back−end and the web interface.

Despite the fact that individuals have been assigned specific tasks on which they should focus their attention, this does not mean that they are only to work on said tasks. All team members are encouraged and expected to assist other members if their assistance or expertise is needed.

V. Required Resources

A. TI SimpleLink^{TM}CC3200 Module

The main node of the system will be built around the TI SimpleLink^{TM}CC3200 Single-Chip Wireless Microcontroller Unit (MCU). This MCU is designed with an ARM ® Cortex ® M4 processor running at 80 MHz, 128 KB of embedded RAM, and peripheral drivers stored in embedded ROM [9]. Additionally, this MCU includes a 4-channel, 12-bit analog to digital converted (ADC), which will be used to convert the signals generated by the sensors into useful information. The processor also contains a variety of peripheral interfaces such as SPI, I2C, UART, SD/MMC, and 4 general purpose timers with pulse width modulation capability. Fig. 2 shows the hardware overview of the MCU.

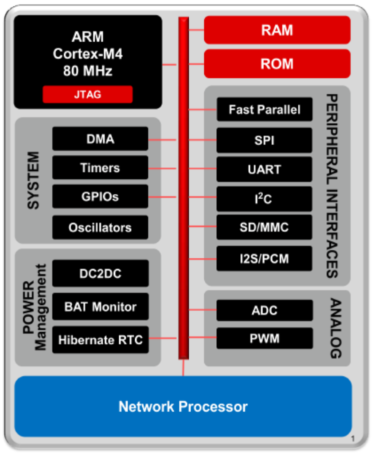


Fig. 2: CC3200 microcontroller hardware overview. [9]

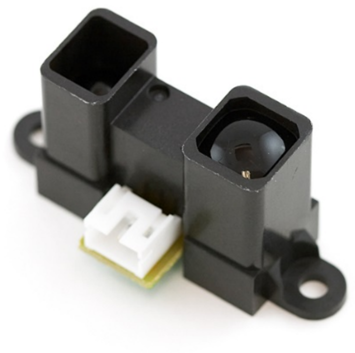


Fig. 3: IR Proximity Sensor [11].

Additionally, the MCU contains a complete Wi-Fi network controller subsystem. The subsystem is capable of communicating under the 802.11 b/g/n protocols. The subsystem is implemented with a dedicated ARM MCU, an integrated TCP/IP stack capable of up to 8 simultaneous socket connections, 802.11 wireless radio and Baseband, and Medium Access Controller.

B. *XBee-PRO® 900HP*

The XBee-PRO® 900HP is a 900 MHz RF Module that communicates using the 802.15.4 ZigBee protocol. This module is incredibly robust, and a complete list of features can be found in the datasheet [10]. The following is a brief list of features that made the module an attractive choice.

- The module contains an embedded ARM Cortex-M3 EFM32G230 processor.
- The frequency range of the module is 902-928 MHz.
- The data rate is 10 Kbps or 200 Kbps (depending on the distance). The data rate of 200 Kbps is achievable up to an outdoor line-of-sight range of 6.5 km. The outdoor line-of-sight range is 14 km for a data rate of 10 Kbps.
- The operation voltage is 2.1−3.6 VDC.
- The current for the transmit, receive, and sleep modes are 215 mA, 29 mA, and 2.5 μA, respectively.
- The supported data interfaces are UART and SPI.
- The module has 4 10-bit ADC inputs, perfect for handling the inputs from the IR sensors on the slave boards.
- The networking topologies include DigiMesh, Repeater, Point−to−Point, Point−to−Multipoint, and Peer−to−Peer.

C. *Infrared Sensor*

An infrared sensor (IR) will be used for detecting objects within a 5 foot distance of each slave node. The exact sensor that will be used is shown in figure 3. The IR sensor sends out a light signal within the infrared frequency range. When an object is within 5 feet or less the light will be reflected back to the device where it will be converted to a voltage signal. An analog output of 2.8 V to 0.4 V is generated with relation to the distance of the object; 0.4 V for objects at a distance of around 5 feet, and 2.8 V for objects within close proximity. This voltage signal will then be converted to a digital signal through an analog to digital converter on the slave device. This digital signal will be used to determine whether or not a particular parking space is vacant or occupied.

D. *Information Delivery*

Once availability has been identified it is important to display the information to a user. As described above, some systems use monitors to display numbers. Other techniques include webpage browsing, email, and text messaging. In Catherine Wah's paper, drivers are notified by SMS and VoiceXML [12]. VoiceXML is an Interactive Voice Response system that connects a caller to a database that delivers information to the driver through a series of voice prompts.

E. *Server*

The server that will be acting as our web host, file sharing, and data processing will be running Ubuntu 12.04 Server Edition. The server will have an Apache Server backend, MySql as the database implementation, and PHP for basic web hosting and web page programming capabilities. The server's domain is p4i.ddns.net. The server can be accessed through ssh telnet applications for authorized members. The web hosted pages can be accessed by the public through the server's domain name. The server application code will be written in C, which will process the master node's data transmissions. Other packages that are installed on the server include: git, openssh, perl, python, samba, and webmin.

VI. Testing and Risk Assessment

A. *Testing*

As is true for all engineering, testing is an integral element of the process. As such we plan to test consistently as the project is being developed. For the embedded software elements of the project, testing will be facilitated through extensive use of the TI CC3200 Launchpad development board. This system will also be where all of the development of the software will take place. Additionally, it will be used for early-stage system deployment and proof-of-concept. For the PCB, and specifically the power systems, testing will be done with the use of schematic simulation tools such as PSpice. Doing so will help us to quickly determine aspects of the project that may not work, as well as help with the overall development of the project. The web interface will be tested early via a database of test data and a walking skeleton of a front-end web client. Additionally, the server itself will be stress tested to determine what type of traffic load the server can take before reliability of the data becomes an issue.

B. *Risk Assessment and Mitigation*

Due to the fact that this project is based on a system that will be implemented in an outdoor setting, there are several distinct risks that are involved, the potential for exposure to the elements being a primary concern. In order to mitigate the environmental effects that the project may experience, the system will be placed in a plexiglass enclosure. This enclosure will be designed such that it is completely waterproof. Plexiglass was chosen for the construction of the enclosure due to the fact that it will not prohibit the signals from propagating from the XBee radios or the Wi-Fi module.

Another potential risk that must be addressed is the possibility of the PCBs not functioning as we expect them to. This scenario is of particular importance due to the fact that the PCBs are the cornerstones of the project. Although every member of the team has had some experience in PCB design, the collective total is minimal. As such, if conditions are such that the PCBs are not functioning correctly and there is not enough time to resolve the issue, the TI CC3200 Launchpad will be used for the demonstration of the system. However, PCB issues, if any, should arise fairly quickly and early in the development of the project, and should be found and resolved with adequate testing.

VII. Results

The sections that follow detail the results of the engineering process for the proposed system. The discussion begins with an overview of the designed hardware including the Master and Slave nodes in section VIII. Following this section, a discussion on the communications systems in the system is presented in IX. Next, the results are presented for the data processing element of the system in X. Finally, the results of the user interface is presented in XI.

VIII. Schematic, PCB Layout, and Assembly

A. *Schematics*

1) Slave Node Schematic: With designing the schematic for the slave node, several key components needed to be placed on the first iteration, see list for requirements.

1) XBee Radio
2) 4 Ultrasonic Sensors
3) Voltage Regulators
4) Source Voltage

The initial design was built intended to use a 9V supply. This source voltage was selected due to the availability of 9 volt batteries. Once testing began the fist iteration results showed that the 9V battery depleted at an exponential rate. Measurements were taken showing that the system drew $\tilde{1}30$ mA during peak transmit time with four sensors attached, and $\tilde{3}2$ mA during low power sleep state. A 9V Alkaline battery only provides 310mAh's of current. This is not sufficient enough to power the system over long periods of time. After this verification a new supply was selected at 5V rated for 16,000mAh. This provided sufficient current and voltage to the system. The first version had three low-dropout regulators (LDO) supplying various parts of the board. The LDO takes a supply voltage and then steps it down to an acceptable voltage for various components. The XBee processor requires a supply voltage of 3.3V. The ultrasonic sensors have a supply voltage that ranges from 2.5-5.5V. 5 Volts was selected for the sensors. This provides grater accuracy and longer range of detection. In the last revision of the board the 5V LDO was removed. With the supply battery from 9V to 5V this LDO was no longer needed. An additional 2.5V LDO was added through further testing. Upon initial testing with the sensors the data being sent was not consistent with the data sheet. Contacting technical support verified that a 2.5V reference was needed to be connected to the Xbee Vref-ADC pin. Once this addition was made the sensors behaved and desired.

One of the goals the project was to meet was low power consumption. This was a major factor for selecting various components. With designing the board one key power consumer was the sensors. Keeping them powered on at all times was an issue. The first version was to program 4 GPIO pins to power the sensors. When you wanted to take a measurement from one of the sensors the idea was to wake it up with a GPIO line which provides 3.1V, and then take a measurement by sending a trigger signal. The trigger signal is sent before a measurement is taken, and alerts that sensor that this is about to take place. This process requires 8 GPIO pins changing voltage values during a measurement cycle. When testing began several issues arose and modifications where needed. The first issue discovered was with the XBee processor. Once the XBee was programmed the GPIO lines voltage values could not be changed. For example if a line was programmed to be high then it could not be reprogrammed to low once the program was uploaded to the processor. With this verification new sensors had to be selected due to not being able to send the trigger signal. New sensors where selected that required no trigger signal. If the sensor is receiving power then it constantly sends measurement values to the processor. This resolves not being able to reprogram the XBee GPIO pins. The second issue was not being able to turn the sensors on and off. To fix this issue the XBee sleep pin was used. One main reason the XBee was selected for the slave board was being able to put the processor into a deep sleep. This was ideal for saving power with programming the processor to sleep for 1 minute, then wake up to take measurements, transmit data, and return to a deep sleep state.

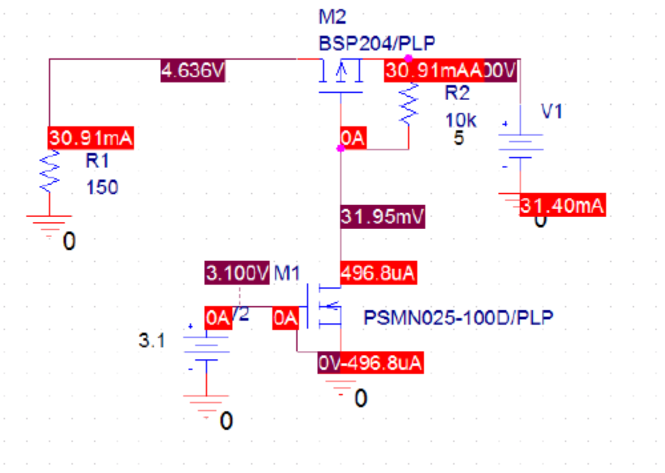


Fig. 4: Transistor Load Switch.

The sleep pin was selected for turning the sensors on and off. With using the sleep pin for powering the sensors presented another issue. Could the XBee provide enough current to power 4 sensors. To resolve this issue a simple transistor load switch was designed. A two transistor IC was selected for each sensor. The IC contained one NMOS and one PMOS transistor. The PMOS acted as an on and off switch for the NMOS transistor. The sleep pin from the XBee was selected as the gate voltage for this transistor. When the sleep pin was set low it biased the PMOS transistor to be off which also caused the NMOS transistor to be biased off. When the sensors needed to be turned on the sleep pin was set high by waking up the processor. Once this value was set high it biased the PMOS to the on state which then biased the NMOS transistor to be on as well. This allowed the source terminal to pass the supply voltage of 5V to the drain of the NMOS transistor, see Fig. 4. This then powered the sensors to the on state where measurements could be taken. This resolved the XBee pin issue and also allowed the sensors to be powered with 5V which is ideal for measurements.

In addition to the transistor switch a buffer IC was added on the output of the NMOS transistor. This provided a higher current to the sensors which increased the power supply. On the final iteration of the PCB zenor diodes where added on the input to ADC's. During testing the output of one sensors had a voltage spike which destroyed parts of the ADC's of the Xbee processor. The diodes added at the input would be be forward biased once the voltage reached 1.8V providing a path to ground and shorting out the input. This would protect the Xbee if the sensors produced any voltage spikes. See XV for schematic design.

In the proposed section the project was originally intended to use infrared (IR) sensors. Upon testing with the first iteration of the slave node, the IR sensors verified to work incorrectly. Further testing verified that the sensors only detected objects two feet from the slave node. This was not ideal and further revision was required. In addition to short range detection the IR sensors worked poorly in outdoor environments. Upon analysis, ultrasonic sensors were selected and replaced the IR sensors. Ultrasonic sensors proved to be more reliable and worked accurately up to seven meters from the slave node which was desired. In addition the ultrasonic sensors worked in outdoor environments which was also desired. The sensor selected was LV MaxSonar EZ Series. The sensor inputs and outputs that were used are as follows. Vcc pin was used for powering the sensors with a 5V supply. The output of the sensor being an analog signal was connected to the ADC input of the Xbee. The sensors output ranged between $9mV$ per inch and $108mV$ per foot [13]. Upon final testing the sensors worked as desired.

Fig. 5: Ultrasonic Sensor [13]

2) Master Node Schematic: With designing the master node several requirements where needed, see list for requirements.

1) XBee Radio
2) TI CC3200 Processor
3) USB Interface
4) Voltage Regulators
5) Source Voltage

The TI CC3200 processor contains 64 pins, ranging from supply inputs, GPIO pins, and pins that need specific multiplexing. To aid in the design the CC3200, and CC3200 launchpad data sheets were used [8]. These data sheets provided information and designs for the USB interface, and multiplexing of the pins. With parts of these designs implemented into the master node additional modifications were made. The TI launchpad contained more products on the PCB than the project needed. Many of the products were removed to decrease the footprint of the master node. In addition to the design, an Xbee processor was added to the master node. This processor received data sent from each of the slave nodes and then was sent over. See XVI for schematic design of master node.

B. PCB Layout

The total time spent on the PCB layout was 42 hours split as follows: 14 hours for the slave node, 27 hours for the master, and 1 hour for the interface board. Time spent creating the footprints to be used during layout was not recorded. While we did not make an estimate on the total hours it would take, we were expecting the layouts for both our master and slave nodes to be finished by the end of June. Because of the need of finalizing the schematic, our first slave node layout was not sent to be fabricated until the beginning of July. Our first, and only, iteration of the master node was not sent out until October. The reasons for the delay in the master node layout will be described later on. Since we were unsure of the efficacy of the master node, we also created a third board which would interface the XBee to the TI cc3200 Launchpad. This connection functioned as our master node for prototyping.

1) Slave Node Layout: The fabrication of all of the slave node boards were done through Circuit Graphics in Salt Lake City, Utah. This company produces the boards in about five days, and, following certain specifications, charged the least amount of money per board. Having the fabrication company close allowed us to obtain the boards quickly thus allowing quick assembly and testing. This proved beneficial as we went through four fabricated versions of the slave node.

The first iteration of our slave node PCB board came from the fourth version of our slave node schematic. When we received this first iteration, we discovered a few problems and potential updates for the next iteration:

- The holes for the headers which connected the IR sensors to the board were incorrectly spaced.
- It was hard to remember which pins on the headers connected to the IR sensor pins.
- The word ”PFI” was smaller than expected, and had plenty of space to have the size increased.
- The spacing of components could be increased to use more of the board space, and to make placement easier.

In the second iteration, which matched the fifth version of the schematic, we made the electrical updates necessary as well as fixing those problems found in the last iteration. We fixed the spacing of the header holes, spaced components a little better, increased the size of ”PFI,” and added letters to the necessary connections for the sensors. There was not anything wrong with this board, nor with the remaining iterations. Most of the updates came from changes in the schematic. One that was not related to the schematic was learned meeting with Jon Davies concerning the master node: a different font choice for component labels would create labels that were easier to read. The third iteration of our slave node layout matched the ninth version of the schematic, the fourth iteration of our slave node, which was not sent for fabrication, matched the twelfth version of the schematic, and the fifth, and final iteration, of our slave node match the thirteenth, and final, version of the schematic. Fig. 6 is the final fabricated slave node.

2) Master Node Layout: The master node layout proved fairly difficult. Most of this was from our inexperience with antenna traces, and multi-layered board design. Once the schematic was completed, placing the components on the board was fairly easy though there were several paths that were difficult to line up. Getting the connections to the inside

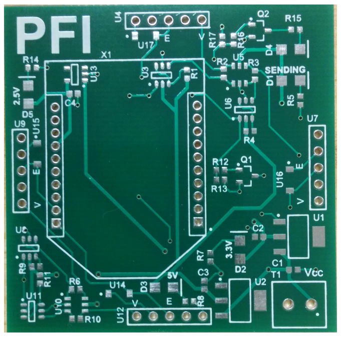


Fig. 6: Front of Fabricated slave Node.

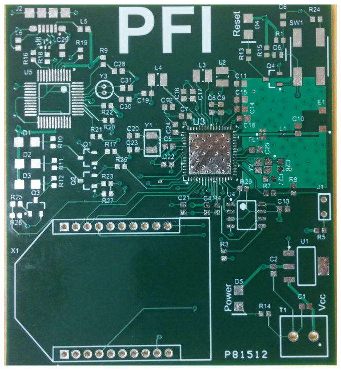


Fig. 7: Front of Fabricated Master Node.

layers of the board, which were a ground and our main power supply, also provided some confusion and time usage. We then got in contact, through email, with an individual, Michael Hollenbeck, who was willing to help us learn more about antenna tracing, and provide correction where needed. We got fairly far along in the process with this person; however, as we approached the final few questions, we lost touch with him. Luckily, we were able to get in contact with someone local, Jon Davies, who was also willing to help us. We met with him, and he provided some key information about the ground plane around the antenna trace and how to properly connect the component pads to the inside layers. We made a few updates as recommended by him, and then met with him to go through those changes. Following a few minor changes, we felt the board was ready to be made. We had the board fabricated through Advanced Circuits since there process could handle the pitch of the pins on the cc3200. They also had a student deal where only one board had to be ordered for a discounted price instead of three or more boards. Without this discount, the price would likely have prevented us from having this board fabricated. Time and financial constraints only allowed for one iteration of the master node board to be made (see Fig. 7).

As we were assembling the board, we found a few errors in the footprints.

- L1, the inductor on the RF line was an incorrect size. We were looking for an 0402 in imperial, but we accidentally got an 0402 in metric which is 01005 imperial. We wondered how this would affect the quality

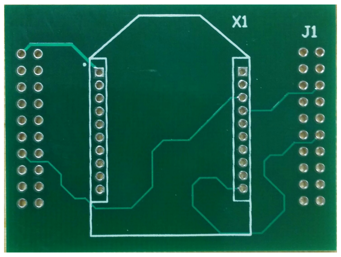


Fig. 8: Front of Fabricated Interface Board.

of our signal.

- The hole sizes for Y3 and J1 were smaller than the necessary sizes though we were able to force J1 into its holes.
- The micro-USB connector required four small holes to assist in a stronger connection to the board, but the pad layout on the data sheet did not make these holes very noticeable so they were not made.

If time and finances had allowed, these errors would have been fixed in the next iteration.

3) Interface Board Layout: Since we were concerned about master node working correctly, we decided to make a simple interface board that connected the XBee board to the TI cc3200 launchpad through the use of headers rather than wires. This allowed for a cleaner look if we needed to use the TI Lauchpad instead of our master node. We already knew that these connections would work so we only needed one iteration of this board. This board was be fabricated through Circuit Graphics. The most difficult part of the layout for this board was to keep in mind that the XBee board needed to be placed in a specific direction so that its antenna was not directly over the TI launchpad antenna(see Fig. 8). Upon receiving the interface board, only two optional ideas for changes were noticed: putting ”PFI” somewhere on the board, and a few LEDs that would light up while the XBee was powered and while it was sending or receiving data.

C. Board Assembly

Both the slave and master boards were soldered by hand. The senior project lab contained decent soldering stations as well as an Amscope Stereo Microscope. The microscope was an invaluable resource in aiding the assembly of the boards. An alcohol based flux was used to mitigate the amount of cleanup the board would need after parts were attached. The solder wire used was a thinner gauge (18-22) which proved useful for some of the smaller parts needed on the master board. Four slave boards and one master board was assembled. The first slave board only had a sensor header installed just to test the functionality of the XBee programming. The board ended up being scrapped since we made an error in the pin assignment. The second board assembled included spots for current buffers but zero ohm resistors were used since the current buffers were not necessary. In the end our third and fourth slave boards used zener diodes to stabilize the output from the sensors. Other changes that were not reflected in the layout, but were solved through soldering was bypassing the 5V LDO. The 9V battery did not provide enough current and was replaced by a 5V battery. Since we no longer had to step the voltage down, we had to solder a line past the 5V LDO. Some of the issues found with the assembly were some parts not solder correctly to the pads and some of the zener diodes being reversed.

Jon Davis had recommended buying a stencil for the master board since there were many components. Stencils are generally expensive, but through some research online, we were able to find a website which provides polyimide stencils at a significantly cheaper price with the understanding that the stencil is not guaranteed for a large amount of uses. None of our team members have had experience using stencils so we asked around, and were able to find a fellow student to help us, Steve Brown. His recommendation was that it would be easier for us to not use the cheap stencil and to solder parts on by hand. Since he was willing to meet with us and help us to solder on the cc3200, we felt we should go with his guidance. In the end the master board was not operational. There were issues with the power and possible problems associated to smaller parts being soldered. Other issues encountered with the soldering process was the fact that the ground plane was exposed for the antenna. This made soldering difficult for some of the smaller parts.

IX. System Communications

The following sections will present the results of the wireless communications deployed in the system. The implementation details as well as the results of testing and deploying a multi-node XBee system are discussed in section IX-A. Then, the implementation and results of transmitting the node information to the server are discussed in section IX-B.

A. Slave Communications

As stated above, communication between the slave nodes and the master node is accomplished via the use of the IEEE 802.15.4 Zigbee protocol. This is implemented via the use of XBee radios from Digi International. The slave node radios are programmed as follows. Each radio has a unique address that is used as the primary identification for the node in which it resides. This address is used to differentiate between radios when processing the data in the master node, and when processing the data in the server. Further, each radio has the address of the master node hard-coded into the firmware so that the radio knows what node to send the data to on a transmit. The radios are each programmed to sleep for a specified duration, then wake up for a sampling period. The radios have four embedded ADCs which are enabled in the firmware as such; these pins can also be used as UART interface, and so ADC operation must be set in the firmware. The input to each ADC is connected to one of the four connected ultrasonic sensors analog output. The output of the ultrasonic sensor is what helps determine whether a given parking stall is taken or empty. The operation of the sensors is defined previously. Finally, the transmit power of the radios is set to the highest level in the firmware, and a data transmission counter is set to 4; this is the number of attempts the radio will make to send the data to the master node before discarding the data.

The operation of the slave node is as follows. The radio sleeps for a specified amount of time, 30 seconds during testing, and then wake up for 30 seconds. In this awake time, the system samples each of the four ultrasonic sensors and latches the values of the ADCs. When a sample is latched, the data is packetized internally by the slave node and sent to the master node. This process repeats a total of 4 times within the awake period. If the master node is not within transmit range, or if for some reason there is not a direct connection from a master to a slave node, each slave node can relay the information to the address of the master node that is programmed in the firmware. If for some reason the transmit fails to complete the number of times specified by the data transmission counter, then the data is discarded.

The results of testing this component of the system are as follows. Initially, the system appeared to operated as expected, with a single node communicating with the master node and data being transmitted, seemingly correctly. However, we soon realized that intermittent data being received by the master node was not correct. The erroneous data was quickly verified through the transmittal of error-packets (EP), packets with all byte values 0xF, to the master. The fact that EPs were being reported by the master node was a definite indicator that the system was not operating correctly. The issue was investigated, and the source of the issue was determined to be an improperly programmed XBee radio. The radio had likely had the firmware corrupted due to a static discharge to the radio; this was a very real problem as the functional integrity of the static mats was unknown.

A second data issue that was encountered was that of a constant sensor data value. This issue was a result of improper board assembly. The board was designed to use a current buffer for each sensor in case the current provided by the radios wake pin was not enough to drive the four transistor switches described in section VIII-A.1. However, in the event that they were not needed, a bypass resistor was placed into the design so the current buffer could be removed safely. The failing board was assembled with both pass-resistors and the current buffers, thereby violating the schematic and producing unspecified behavior.

Once the two previous issues were resolved, testing continued and yet another issue arose. This time, the data being transmitted to the master node was correct, but an incorrect number of packets were arriving to be processed. The cause of this issue was attributed to interference in the lab where the development and a majority of testing took place. We cannot say for certain what caused the interference, but the theory that interference was the cause of the improper operation was validated when the system was removed from the lab and tested; the system resumed working as expected.

After these issues were resolved, the mesh network was successfully deployed with two functional nodes communicating with the master node simultaneously. The communication between the slave nodes and the master node was robust enough that there was no loss of data transmitted even when the nodes were separated by walls and other objects, as was expected. The other slave node in the deployed and tested system operates exactly as specified above, and the results of the testing are similar.

B. Master - Server Communications

The master node is composed of an XBee radio and a TI processor, specifically the CC3200 SimpleLink MCU described previously, embedded into a TI CC3200 Launchpad. A description of the operation of the system follows. The TI CC3200 Launchpad (TI) is mainly responsible of ensuring that the data generated by the slave nodes is correctly transmitted to the server. The TI Is connected to an XBee radio, the coordinator, via GPIO lines and together the two components are responsible for sending the data generated by the slave nodes to the server for further processing. The radio is programmed in a manner similar to the that described in the previous section, except that the ADCs are disabled, and the SPI interface is enabled. The TI is programmed with an application that enables it to pull the data from the radio when an interrupt is asserted, as well as transmit the data to an off-site server for further processing. Additionally, the TI makes use of I/O pin muxing to properly configure the various GPIO lines to behave as a SPI bus.

The coordinator XBee radio operates as follows. When a slave radio enters an awake cycle, the data latched from the ADCs is immediately transmitted to the master node coordinator radio. The master node radio then takes the transmitted data and queues it up for transmission to the TI via a SPI interface. The radio then sets the SPI interrupt signal low, and the process of dumping the data via SPI begins with the radio acting as the SPI slave, and the TI as the master. Along with the data pertaining to the four ADC from the given slave node, the data packet also contains the address of the node that transmitted the data, the size of the data packet being transferred, a checksum for the data, and other data. This data packet is predefined by the radio manufacturer and we are not able to modify it. Additionally, if there is an error in the data transmitted by the slave, or if there is an error with the functionality of the master node radio, the radio itself will produce a packet of data where every byte has the value 0xFF.

The functional purpose of the TI is to accept the data generated by the slave nodes and then transmit it via Wi-Fi to the server. The functionality of the embedded application is presented below. When powered on, the application on the TI first the necessary GPIO lines to act as a SPI bus, and configures peripherals such as the Wi-Fi subsystem and UART to their appropriate states. The application then connects to the wireless network specified in the application code. If the connection cannot be made, an error is reported. Additionally, the application registers an interrupt handler for the active-low SPI interrupt line in the interrupt table. This Interrupt Service Routine (ISR) is responsible for transferring the data from the radio to the TI. After all of these initialization steps have been completed, the application goes into an infinite loop that waits for a flag to be asserted. The assertion of the flag and how the interrupt is executed are described below.

When the SPI interrupt signal from the coordinator radio goes low, The main process of the application is interrupted and the ISR begins execution. Firstly, the SPI peripheral is reset to its default state, the SPI clock is started, and the data containers are allocated. Next, the data from the coordinator radio is clocked into the TI. Afterward, the data is copied to a global data buffer for processing in the ISR. Throughout execution of the ISR, information pertaining to the data packet received is printed to a console via UART for rapid debugging. After the data has been completely offloaded from the radio and stored into the global processing buffer, the data is re-packetized, stripping away data that is not relevant to a sample packet such as the checksum. Additional data pertaining to the ID of the master node that received the data is added, as well as a packet signature to ensure that the data received by the server is a legitimate. Once the data is formatted correctly, it is stored into another global buffer and the ISR done flag is asserted, signaling the completion of the ISR. When the ISR process is complete and execution returns to the main process, the flag will be high and the application executes a server transmission routine to transmit the data to the server. In the server messaging routine a TCP socket is created using the network information specified in the application file. The creation of the TCP socket is abstracted away via the use of API functions and macros provided by Texas Instruments. However, the application must correctly specify the parameters with which to open a TCP socket to the server. Upon a successful connection, the message stored in the global pointer is transmitted, and the routine exits with a success message printed to the console. The flag is deasserted in the main process, and the the application again waits for an interrupt.

The results of the testing are as follows. The system operated as expected when first fully deployed. The system was functionally tested with two slave nodes operating simultaneously with different wake and sleep periods. There was no perceived collisions with data packets from different slave nodes, as was expected. Additionally, the communication between the application and the off-site server had no issues with respect to implementation.

There was a major issue relating to authentication certificates that was encountered when attempting to connect to the guest network at The University of Utah. We quickly realized that we would not be able to get an authentication certificate for our device in a timely manner, so an alternative network solution was employed. The initial solution callef for the system to use a Wi-Fi hotspot provided by the team to connect to the server, however this was impractical, and a separate access point that used the wired connection was deployed for our use.

With respect to the integrity of the data transmitted from the radio the the TI, there was only one minor issue. The data being stored to the buffers was stored in a big-endian format, but in a little-endian ordering. the order of data that was larger than one byte was rearranged, but the total ordering was preserved. This was a source of much confusion when debugging, as the data appeared to be incorrect. The cause of the byte reordering was not found, however the server was capable of resolving this issue, and will be discussed in a later section.

X. Server and Database

The following sections present the results of the server deployment and the database scheme used to represent and store the data. The implementation and results of testing the server are covered in section X-A, while the results if the database are presented in section X-B.

A. Server

The results of implementing the server application for the system are as follows. Throughout the course of the project, three different servers were implemented. The first iteration of the server was a C++ server that ran using the Boost libraries. The second server was developed in C, and was a low-level TCP socket server. The third implementation was a Python server that made use of the SocketServer Python library and a multiprocessing queue. The implementation details and test results for each server will be discussed in the preceding sections. The final server that was deployed and used in the system was the Python server, and justification will be provided.

1) C++ Server: The C++ server implementation made heavy use of the Boost library, specifically the Asynchronous IO (ASIO) library. This language was chosen for the implementation due to prior experience held by the team in developing a server in this language. The server was structured in a server-session model, in which every incoming connection has a new session object created. When a incoming message is detected, an asynchronous callback is executed so as to not interfere with the main server process. This callback is responsible of extracting the data that was transmitted from the socket. The data received was then printed to the console for debugging purposes, and then stored for later processing.

This server implementation was not fully developed due to inconsistencies with the data received during testing. The data that was received by the server did not even closely resemble the data that was being transmitted by the master node. Even when the endian-ness of the system, and the bizarre endian switch discussed in section IX-B, was taken into consideration, the data was not correct. This issue was investigated for an entire week, a reasonable amount of time, at which point an executive decision was made to investigate an alternative server implementation. Luckily, we had already prototyped a C server, which will be discussed in the following section.

2) C Server: The C server was the product of an early research and development task on the workings of a server implemented in a low-level language. Due to the fact that the embedded application was written in C, it was deemed that a C server would be the best implementation. However, this idea was shelved in favor of the C++ server, as we felt confident in the higher-level implementation as detailed above. The server was implemented with a single server that handled all incoming connections. Each connection was assigned a processing thread and allowed to run in its own context. Additionally, each thread was assigned an identification number so it could easily be referenced throughout the server. These identification numbers were stored in an array, and removed when the thread terminated. The connections were serviced by a use of a callback which unloaded the data from the TPC socket and processed the information. The data that was received from this server was correctly formatted with respect to the transmitted packet. However, it soon became clear that this server would not be scalable for a multi-master multi-node system, as we had designed for.

The results of testing the C server are as follows. Despite the fact that the data being transmitted correctly during testing, other errors arose. The most serious of the errors was a segmentation fault that was produced after an extended period of operation. The error would manifest after several minutes of operation. It was determined that this error was caused due to the fact that the thread identification numbers stored in the server, in an array, were never being removed correctly, and thus an out of bounds array access was occurring after a set number of connections. While this was a relatively easy error to fix, it was deemed at this point that this server implementation was simply not capable of scaling as we had intended. A decision was made to research other server implementations that would be better suited to the scalability that our system was designed for. The results of the research resulted in the Python server described below being used in the final implementation of the design.

3) Python Server: The implementation details of the python server are presented below. Similar to the C-server, the Python server consists of a single server that handles incoming messages asynchronously through the use of callbacks. However, the Python server makes use of callback objects, as opposed to a single function that is part of the server object. Additionally, this implementation makes use of a Python Multiprocessing Queue (MPQ). The MPQ is a queue object that can be instantiated in its own process, and so long as its pointer is passed to another process can be accessed in n-many processes. The MPQ was used to queue the incoming message data from the master node. This implementation was chosen due to the ease of scalability offered by the MPQ. A functional description of the server follows.

The server is started via command line. Upon initialization, the MPQ for the server instance is created before the server object is. This allows the MPQ to be in the global scope of the process, and therefore accessible by the server and callback objects. The port and IP address of the host are hard-coded for ease of deployment in the main function,

TABLE I: ADC data interpretation

Data Value	Taken/Empty	Int. Value
data $>$ 0x7b	Empty	0
data $\leq$ 0x7b	Taken	1

and the server object is created. The initialization of the server is as follows. The constructor for the server stores the parameters passed into it, namely the IP address for the server and the port on which to listen for incoming connections. Then, a SocketServer object is created with the port number and IP address, as well as the class name for the callback object. Various parameters are set, such as allowing address reuse, and then the constructor terminates. The server is then launched in a process separate from the main process, and the main process is stalled until termination of the server process, effectively allowing the server to run indefinitely. The server merely runs forever, waiting for a connection. When a connection is detected, it creates an instance of the callback object to handle the incoming communication. There is only a single function declared in the object, as all of the code related to the creation of the object is abstracted away. Upon creation, the singly defined function is executed. The sole purpose of this function is to take the incoming data and append it to the MPQ. Once this is completed, the callback thread terminates. Nearly all of the data processing is accomplished in the MPQ, which is discussed in the following section.

The functionality of the Multiprocessing Queue is as follows. After initialization, the MPQ simply waits for a specified period of time and check whether the queue has any elements to process. If there are none, then the MPQ simply waits and checks again. If there are elements that need to be processed, the following occurs. The top element is removed from the MPQ and the bits are unpacked through the use of the Python Struct library. This library allows the developers to specify a byte ordering of binary data processed in an application. With this, we were able to overcome the endian switching issue that was occurring with the data being offloaded onto the master node. Once the data is unpacked and stored to local variables, the data pertaining to the address of the slave node is analyzed and checked for any erroneous values. If the address is deemed to be erroneous, the entire data set is discarded and the MPQ processes another data set. If the data is valid, the data representing the outputs of the ADCs is processed. The unsigned binary value of each data sample is checked according to Table I, and an integer value of 1 or 0 is appended to a string representing the order of the spots.

The string representation of the ADC data is printed to the console for debugging purposes, as well as the address of the slave node from which this data originated. The data is then stored on to the database described in section **??**, and the process begins again.

The Python server was chosen as the ideal server implementation for several reasons. Firstly, the built-in libraries available to developers allowed us to overcome all of the

TABLE II: Node Data Table

slaveNode	parkingSpots
slave1	0000
slave2	1010
slave3	1100

errors that were present in the previous server implementations. With the SocketServer library, all of the memory management issues of a server were handled by the library. We merely had to implement the server using the correct API implementations. The only major issue that was encountered was that of a server process persisting despite the application receiving a process-kill signal. We determined that this was due to the fact that the signal was not being propagated down to each child process of the main process, and therefore any child process, such as the server process, was not being properly terminated. Once this issue was resolved, the testing of the Python server was completed without any other major functional issue. We believe that this was due to the fact that we had uncovered a majority of the issues that could be encountered with our particular implementation of the server in previous implementations. Upon deployment, the server operated exactly as intended, and was one of the most stable pieces of software in the entire system.

B. Database

The server implements a MySQL database in order to keep track of reported parking lot node data transmitted to the server. Table II illustrates how the slave node data is being stored in the database. Each slaveNode has a unique ID which is generated by the XBee processor. The parkingSpot data is a four bit string representation of the four spots that the slave node is monitoring. A one asserts that the spot is taken, whereas a zero means the spot is open. The appServ application performs an UPDATE statement that, depending on the reporting slaveNode, will update the database with the newly captured parking spot data.

The schema is simple yet effective for the implementation of this system. Once the unique IDs for the XBees were known, the table was populated with default values of '0000' to represent an open parking lot. In order to test the UPDATE statements we had to have the slave nodes reporting information to the appServ application. The application used print statements to validate the parking spot data and this was compared to the values stored in the database tables.

The main issue in testing the database update was correctly populating the string representation of the XBee ID and dynamically updating the tables. The first assumption was that the values were going to be hexadecimal, but instead were actually a number representation. Once this issue was addressed the updates worked as expected.

In addition to the Node Data Table, we also implemented a reservation table. Table III shows how the reservation data is being stored. This table is populated through the User Client website. Once a User has logged in and been authenticated, any open spot can be selected for reservation. The Spaces in the table are a string representation of the open spot selected in the parking lot. The expiration time is a timestamp that is taken at the time of the reservation. This information is passed to the database using an INSERT statement generated from a PHP page. There were no issues found when testing this functionality.

TABLE III: Reservation Table

Spaces	Expiration Time
1A	144976681
1B	144976792
2D	144976814

TABLE IV: User Table

Id	userName	Password
1	user1	pass1
2	user2	pass2

The last database needed for the parking lot system is a User database. This database contains the username and password information for users with access to parking spot reservation. The format of the user table is shown in Table IV. The Id is an incremental unique key to identify users. The userName field is also initialized as a unique key so no two users can share a login name. The Password field in its current form will accept any varchar(40) input.

XI. User Client

A. Main Page

The User Client web application is written with HTML, CSS, PHP, and JavaScript. The application is accessed through the web site and the main parking lot information is displayed in a user friendly table (see Fig. 9). The web page is refreshed every 10 seconds to display the most up-to-date information of the parking lot as stored in the node data table. The main page also displays information regarding open spaces or spot reservation if the user has logged in to the website (see Fig. 10).

The main page performs two separate MySQL queries, as well as a DELETE statement to manage and display the parking lot data and reservation requests. The first query selects all data that is loaded to the Node Data Table (see Table II). This data is loaded to an array for each reporting node. The first index represents the top left, the second index for the top right, the third index for the bottom right, and the fourth index the bottom left parking spot location. If the value at the index contains a '1', then the spot is considered full and an image of a car is displayed, otherwise the function checks for possible reservations.

The second query selects all data that is held in the Reservation Table (see Table III). This data is also stored to arrays, but first the expiration time needs to be checked. By default, the web application is set to expire any reservations after one minute. If the reservation has expired, the entry is deleted from the table. If a parking spot is not full, but is marked as reserved then a 'Reserved' banner is displayed in the spot, otherwise the spot will display the name of the location and be visible in the Reserve or Open Spot table (see Fig. 10).

The main page has a subtle change in formatting depending on whether or not a user has logged in. Fig. 10 is shown when the page has detected an authenticated user. Each spot is a hyperlink for which the user is prompted to accept a reservation. The programming to capture the OnClick event is written with Javascript. The selected spot is posted as a query string parameter which a subsequent PHP page extracts the parameter and performs an INSERT statement to the reservation table (See Table III). If the user has not logged in then instead of having the option to ”Reserve a Spot” the user will see ”Available Spots”.

The first issue that was encountered in programming the web application was correctly populating the parking lot data from the Node Data table. The naming convention that was chosen to represent the spaces did not integrate well with the reported data nodes, so the open spaces had to be hard coded. To solve the issue, the data from each node was loaded to their own array.

Another issue with the web application was that the cars were not loading to the correct parking spot. The problem was that the server application was saving the node data in descending order, i.e. Spot 4, Spot 3, Spot 2, Spot 1, and the web application was reporting the spots in ascending order. To fix this issue we reversed the server application programming to be aligned with the ascending spot order.

Additional issues with the web application was handling parking spot reservation. The first was being able to identify if a user was logged in. The login page was capturing session ID data, but it was not being passed back to the main page. In order to resolve the session data, we had to have the session creation performed in the main page. Once the session was created in the main page, the login page could modify the existing session and store the user name and password. The other issue was determining how to prompt the user for which spot they wished to reserve. The first approach was to pop a second page that the user could select from a drop down box which spot they would like. The outcome was a clunky page that didn't work properly and was not very user friendly. To make a more streamline selection function, JavaScript coding was injected to watch for OnClick events. The event identifier and hyperlink was populated dynamically based on the open spot name. Once this solution was in place, the reservation functionality worked as intended.

B. Login Page

The login page is accessed from the main page of the web application. The page simply prompts the user for a username and password. The application queries the User table (See Table IV) and if a match is found assigns a session ID. This session ID acts much like a COOKIE, where it defines some information about the user. This application just stores the username and password of the user if a successful login

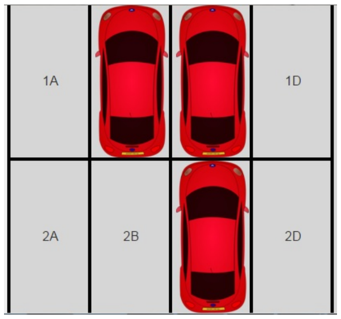
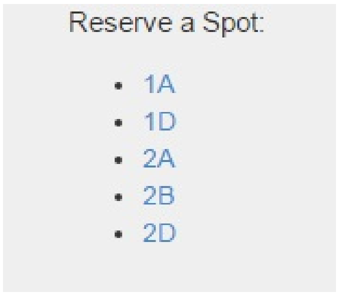


Fig. 9: Parking Lot information display table.



Reserve a Spot:

- 1A
- 1D
- 2A
- 2B
- 2D

Fig. 10: Parking Lot information display table.

occurs. The session data is used in the main page (Section XI-A).

As described in Section XI-A, there were issues with passing session data back and forth between the main and login web pages. Another issue was deciding between using cookies or session data. In the end we went with session data due to issues experienced with users not having cookies enabled in their web browser.

XII. Whole System Analysis

As a whole, the implemented system behaved nearly as expected when deployed. The communications between the slave nodes and the master nodes did not exhibit any errors. The ultrasonic sensors used did exhibit some undesired behavior, however we believe that his was due to the system being deployed in a noisy environment and not in an ideal outdoor scenario. Additionally, the frequency of the erroneous behaviour was such that it did not affect the overall reliability of the system. The communications between the master node and the server did not exhibit any erroneous behavior, and all data that was reported by the master node was correctly reported by the server, signaling a successful data transfer. The web interface also did not exhibit any errors in its output of the data from the database, thereby proving that the data being stored to and retreived from the database was consistent. With all of the separate systems working correctly when deployed as a whole, we conclude that the deployment of the system was successful.

XIII. Summary

Finding an available parking space can be difficult in large lots. The difficulty scales up when there are numerous individuals vying for the same space, and when time constrictions are in place. Through the use of ultrasonic sensors, XBee radios using the ZigBee protocol, and Wi-Fi communication, our solution detects open spots, communicates this information to a server, and the data is processed and displayed in a user-friendly manner. This system has the potential to mitigate the annoyance and hassle associated with finding a suitable parking space.

XIV. Acknowledgements

We would like to thank Dr. Ken Stevens for his guidance throughout the year. Without his encouragement and persistent check-ins this project may have ended poorly. We also thank Alex Ferro for his indispensable help debugging esoteric circuit issues. We thank our families for putting up with our late nights working on the project, and for helping us deal with the stresses of a project such as this. For help understanding the RF line on the master node layout, we thank Jon Davies, and Michael Hollenbeck. We also thank Steve Brown for his help in soldering the cc3200 chip to the master node.

References

[1] D. L. Davenport and D. A. Beach. (2009, Jan) Crash Summary 2008. UtahCrashSummary2008_001.pdf. [Online]. Available: http://publicsafety.utah.gov/highwaysafety/documents/UtahCrashSummary2008_001.pdf

[2] K. D. Squires, D. A. Beach, and G. D. Mower. (2014, Jan) 2013 Crash Summary. 2013UtahCrashSummary.pdf. [Online]. Available: http://publicsafety.utah.gov/highwaysafety/documents/2013UtahCrashSummary.pdf

[3] T. O. of Budget and I. Analysis. (2008, Oct) Headcount Enrollment by Academic Level, Gender, and Ethnicity 2008 Autumn Semester Census. ss0809A02.pdf. [Online]. Available: http://www.obia.utah.edu/ia/stat/2008-2009/ss0809A02.pdf

[4] ——. (2014, Oct) Headcount Enrollment by Academic Level, Gender, and Ethnicity 2014 Autumn Semester Census. ss1415A02.pdf. [Online]. Available: http://www.obia.utah.edu/ia/stat/2014-2015/ss1415A02.pdf

[5] ——. (2014, Oct) Common Data Sheet. CDS_2014-2015.pdf. [Online]. Available: http://www.obia.utah.edu/ia/cds/2014-2015/CDS_2014-2015.pdf

[6] A. Shanbhag, G. Ames, and P. Aaronson, "Real Time Detection of Parking Space Availability," Feb. 22 2011, uS Patent 7,893,847. [Online]. Available: https://www.google.com/patents/US7893847

[7] R. Krivacic, R. Hoover, E. Isaacs, and J. Glasnapp, "Computer-Implemented System And Method For Spontaneously Identifying And Directing Users To Available Parking Spaces," Sep. 4 2014, uS Patent App. 13/783,070. [Online]. Available: https://www.google.com/patents/US20140249742

[8] T. Instruments. (2015, Feb) AN123 Breaking the 400-Node ZigBee® Network Barrier With TIs ZigBee SoC and Z-Stack™Software. swra427.pdf. [Online]. Available: http://www.ti.com/lit/an/swra427c/swra427c.pdf

[9] ——. (2015, Feb) CC3200 SimpleLink™Wi-Fi® and Internet-of-Things Solution, a Single-Chip Wireless MCU. c3200.pdf. [Online]. Available: http://www.ti.com/lit/ds/symlink/cc3200.pdf

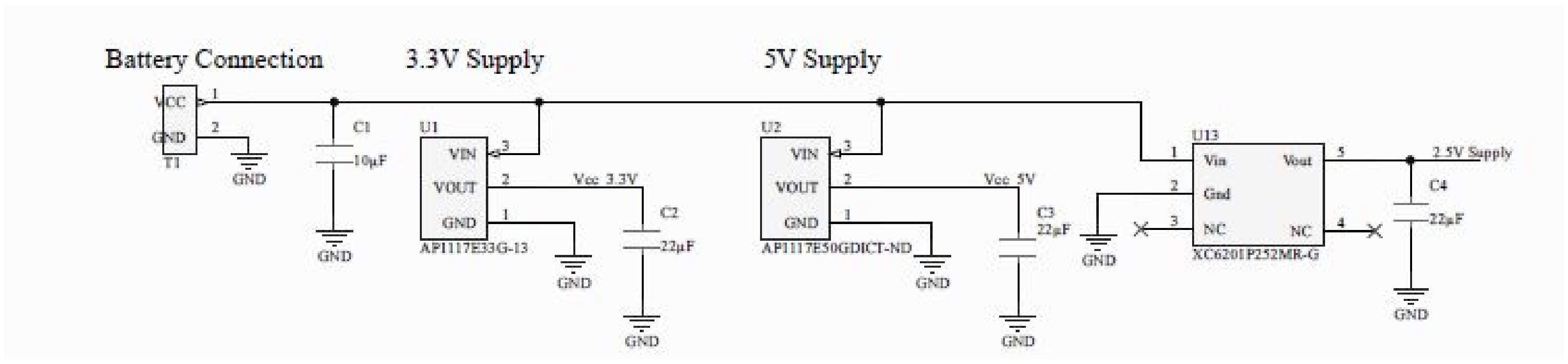
[10] Digi. XBee-PRO® 900HP 900 MHz RF Module. ds_xbeepro900hp.pdf. [Online]. Available: http://www.digi.com/pdf/ds_xbeepro900hp.pdf

[11] Sparkfun Electronics. Infrared Proximity Sensor Long Range. [Online]. Available: https://www.sparkfun.com/products/8958

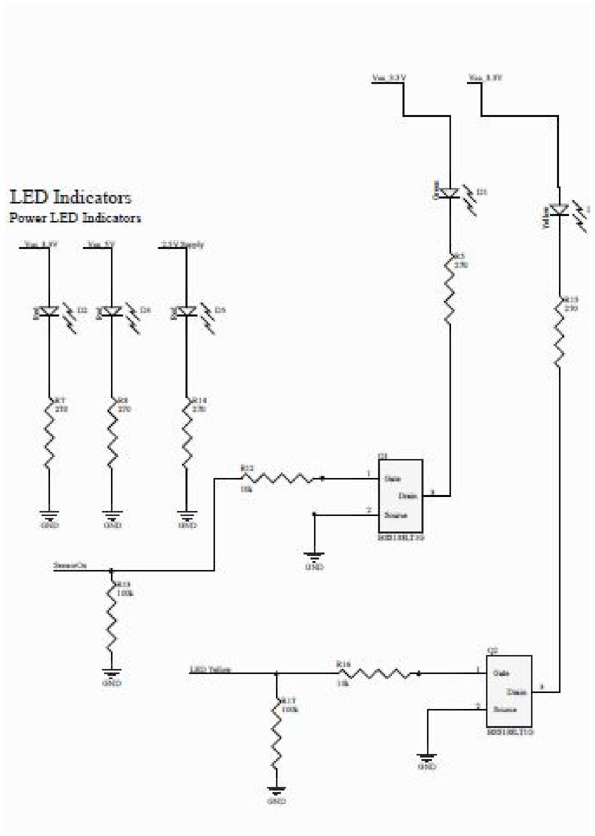
[12] C. Wah, "Parking Space Vacancy Monitoring," *Projects in Vision and Learning*, 2009.

[13] LV–MaxSonar–EZ. High Performance Sonar Range Finder. [Online]. Available: http://www.maxbotix.com/documents/LV–MaxSonar–EZ_Datasheet.pdf

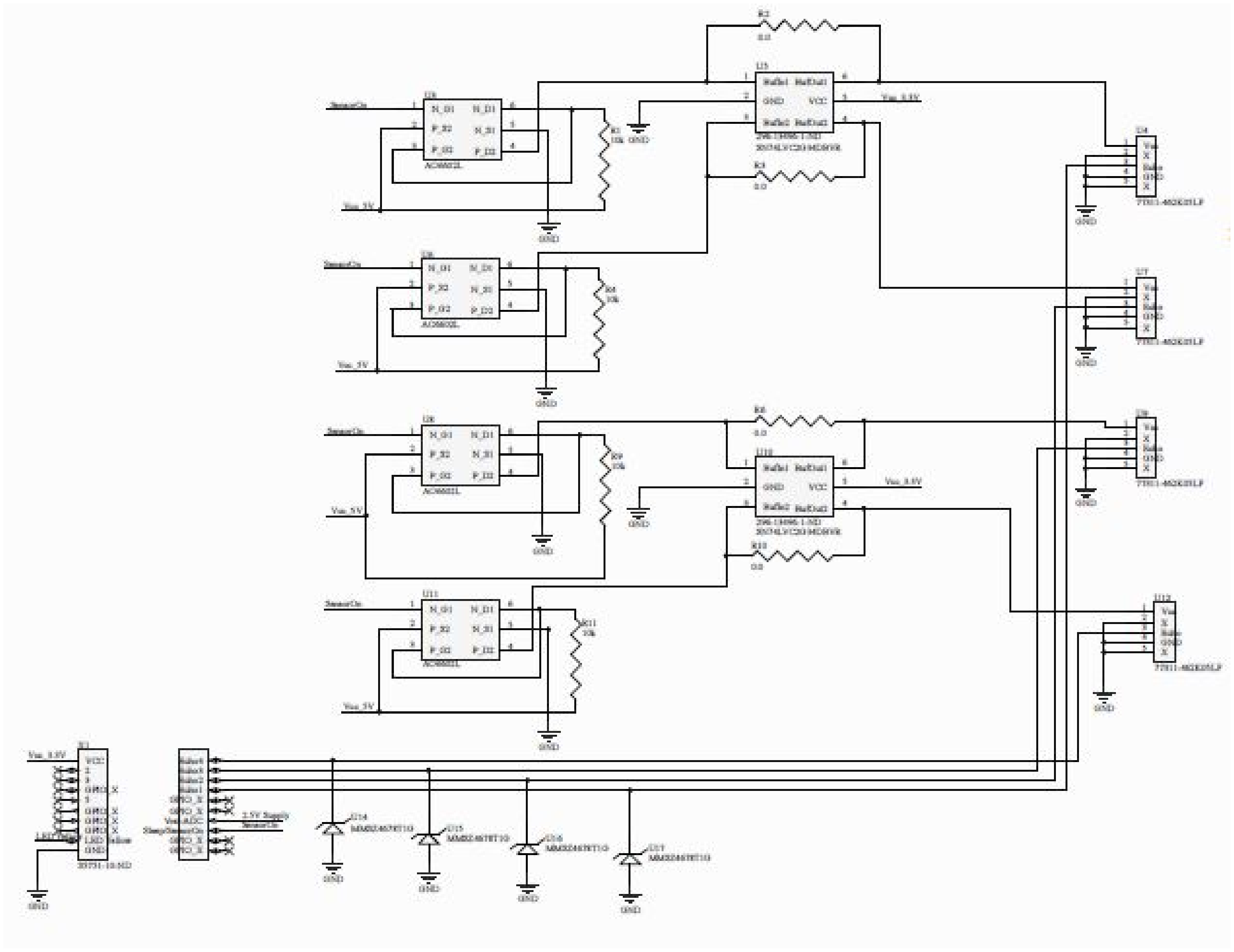
XV. APPENDIX A



Slave Node Power Supply.

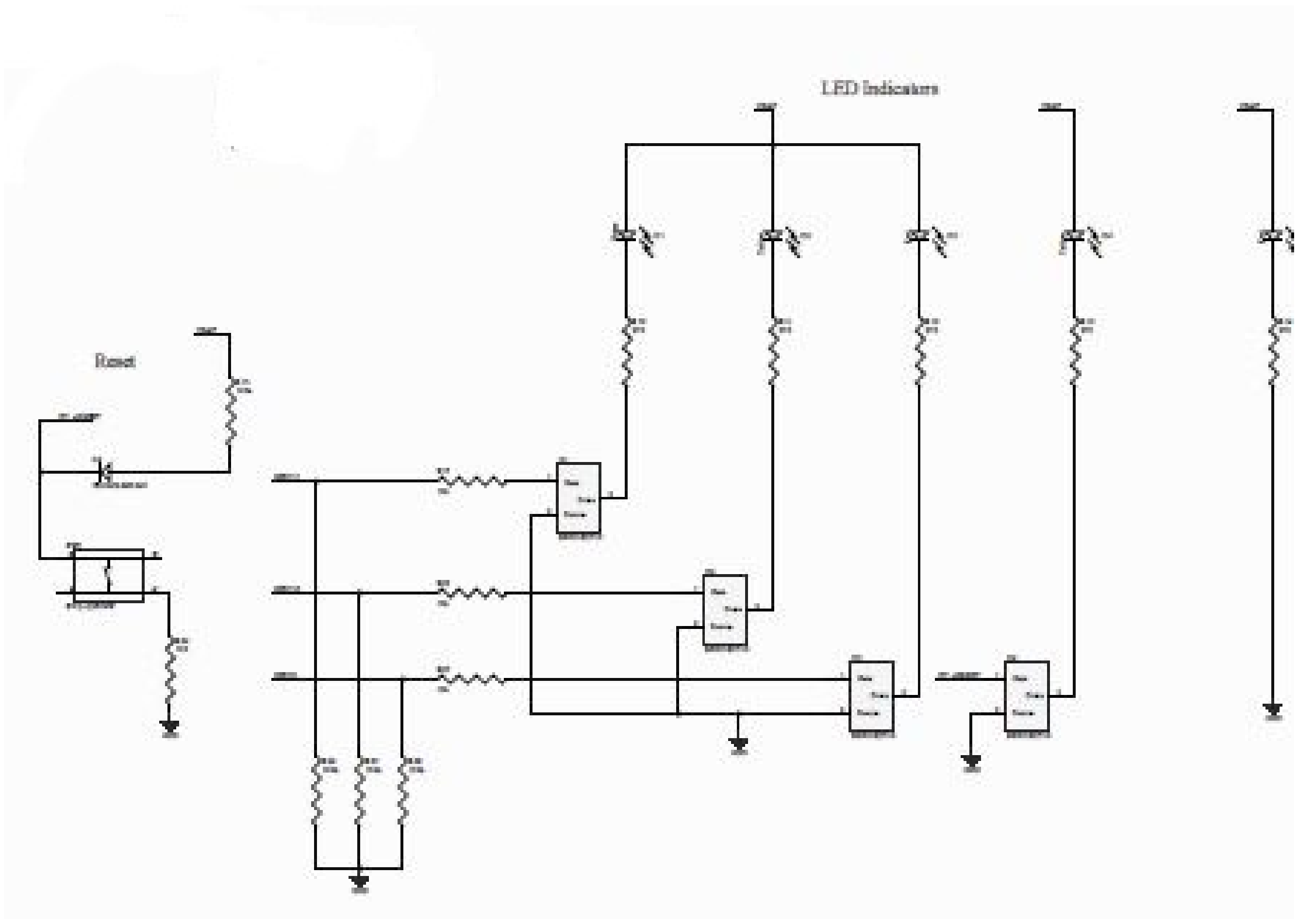


Xbee and Sensors.

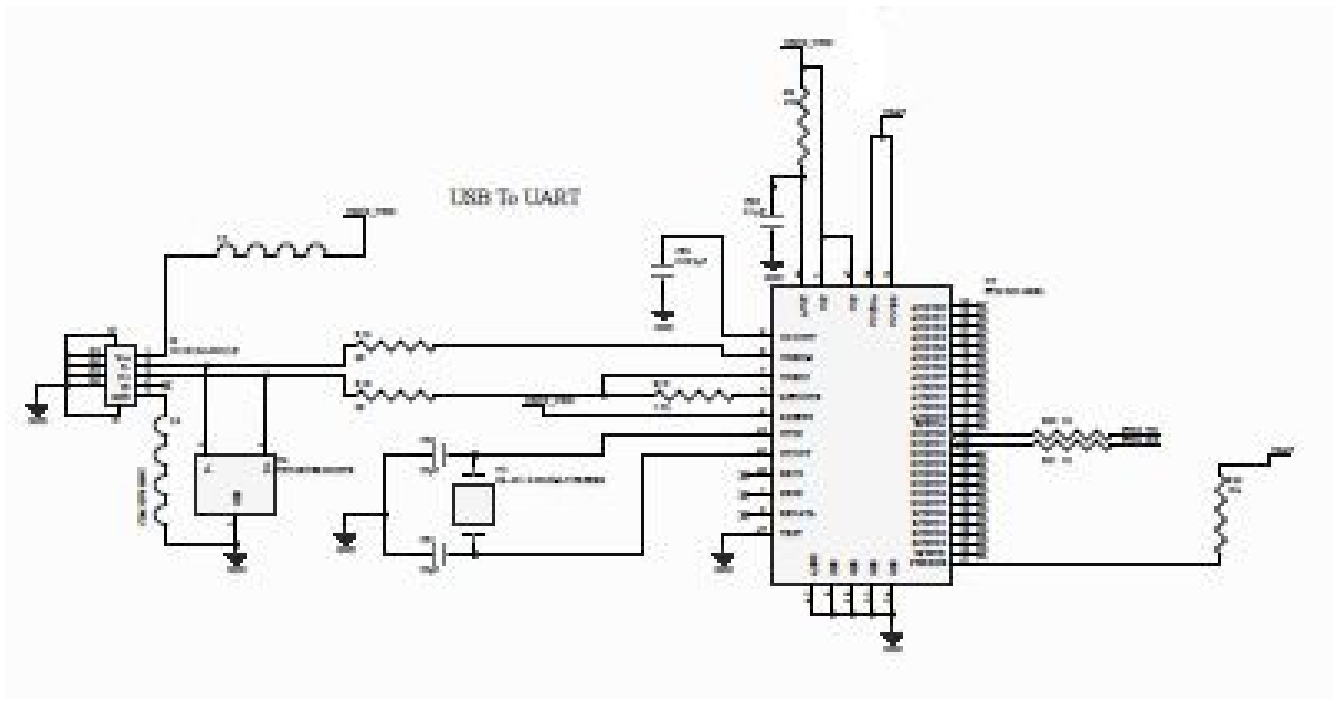


Xbee and Sensors.

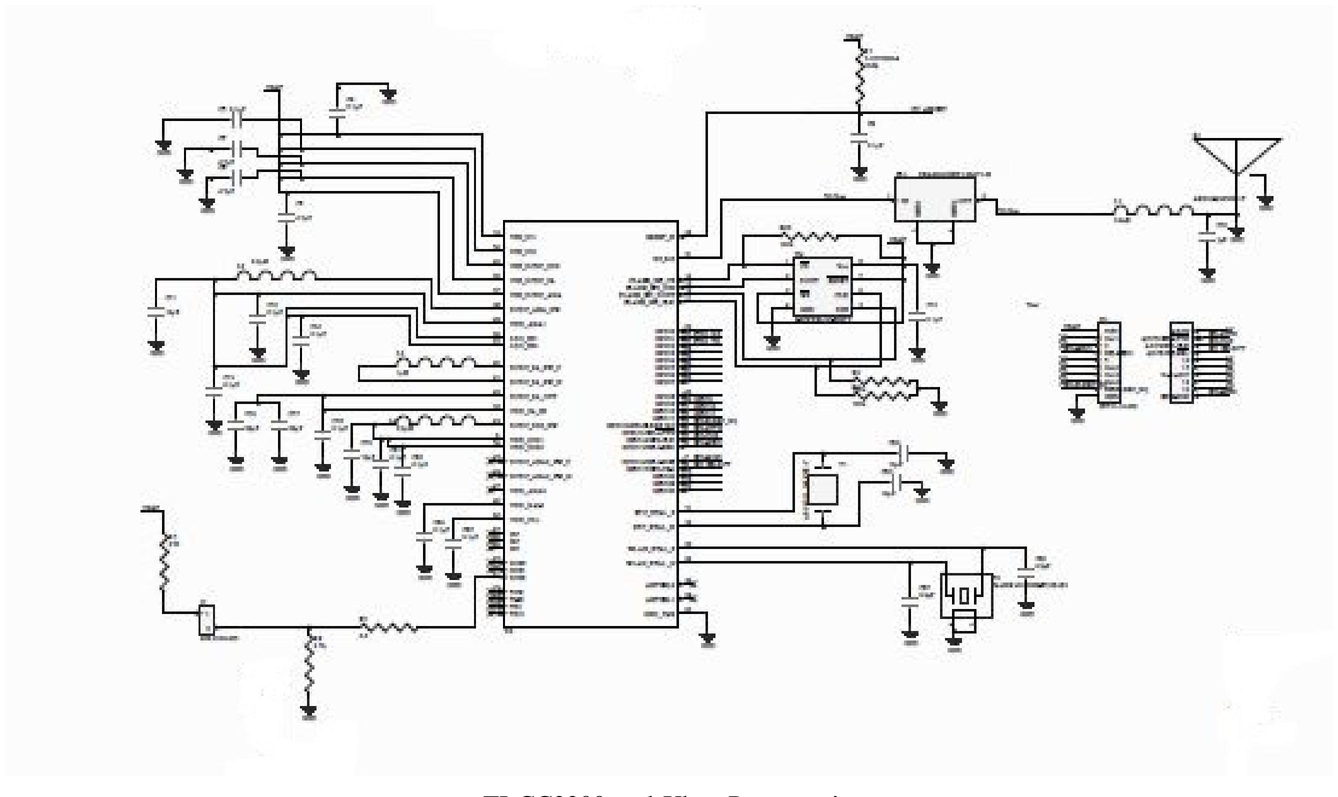
XVI. APPENDIX B



LED's.



USB to UART.



TI CC3200 and Xbee Processor’s.