Backup Buddy – Wireless Backup Camera for Motor Vehicles

Dylan Ortiz, Coleman Rogers, Luca Silvester, and Zachary Slakoff

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — Most modern cars come equipped with back up cameras, however consumers looking to get a used car don't have as much a say in whether this feature is included, even more so if the car is an older model. Aftermarket backup cameras exist, but most are hard wired and require the user to run cables throughout their vehicle, something that is both inconvenient and unnecessary. Backup Buddy looks to solve both of these issues by offering a backup camera solution for everyone, that involves minimal installation and works wirelessly through an Android app.**

***Index Terms* — Accelerometer, Android, Bluetooth low energy, I^{2}C, raspberry pi, UART, ultrasonic sensors.**

I. Introduction

Thanks to many advances in technology the standard options available to people looking to buy new cars has improved in recent years. But going back just a few more years and premium options like a back-up camera were often out of reach for anyone except those who were willing to pay thousands of dollars extra. While many people can get by just fine by looking behind them as they back out of a parking spot, nobody is perfect and there still exists room for error and mistakes can be made. The dangers of backing out are clear and present, according to the National Highway Safety and Traffic Administration, more than 18,000 backup-related injuries occur in the United States each year with more than 200 of these injuries being fatal [1]. An unaware or distracted driver operating a 2000-lb vehicle creates a significant safety hazard for those walking nearby and behind their backing-up vehicle. What if the driver looks away for a second to check their phone? In those few seconds, someone could appear that the driver didn't see initially. Thinking they are still clear they continue to back-up, running the risk of unknowingly hitting the pedestrian. Without a clear view of what's behind them, many drivers will just start to back out slowly. This makes backing-up out of a parking spot a needlessly dangerous guessing game for everyone involved. Fortunately, this potentially dangerous situation can be made safer by providing the driver with more information about their surroundings through the use of a backup camera on the vehicle.

II. System Component Overview

The product we are striving to deliver on this project is a small attachment to the rear of a motor vehicle, around the license plate. This device would be capable of streaming a video feed of the back of the vehicle with minimal latency to the user's smartphone. In the app the user can also see information from a series of ultrasonic sensors built-in to the device that provide precise distances to objects behind them, and can provide audible warning through the app, should the distance between the vehicle and an obstruction behind it get too close. Once a user has successfully pulled out of the parking spot and begins to drive away, an on-board accelerometer would register the significant increase in positive acceleration and trigger a shutdown of the video stream, dropping the system into a low-power state while it waits to begin streaming again.

The enclosure was designed using 3D modeling software and printed on a 3D printer as 2 pieces. The rear piece contains all of the hardware and mounting points for the screws on the rear of the car. The front piece of the enclosure is used to hold everything inside. The enclosure holds 3 ultrasonic sensors on the sides and top of the enclosure, with the bottom section housing the PCB, where the microcontroller and accelerometer reside, as well as the camera and raspberry pi for the video stream.

From these features we decided on, as well as the logistics of creating them, the following table shows the requirements specifications for our design.

	Hardware Performance
1	The system will draw no more than 12V from its power source
2	The system will weigh less than 10 pounds
3	The face of the system will take up a space no larger than 16in x 10in
4	The system will be able to accommodate up to 2A of current draw under load
5	The size of the PCB shall be no larger than 2.5in x 5in
	Software Performance
1	The Android application will make an audible tone as well as provide a visual warning on screen, to alert the driver when the distance to an obstruction falls within an unsafe range of values.
2	The system will be able to detect obstructions that are behind the vehicle within a field of view of 4 feet and the size of 1 cubic foot.
3	Using an accelerometer, the system will detect once the car is moving forward and when the car reaches a speed of faster than 10mph will turn off the rear facing camera

4	At all times that the car is in reverse, the camera feed will be sent to the Android application
5	The video feed of the rear facing camera will have a framerate of at least 15 fps at any given time

Fig. 1. Requirement Specifications

The following state machine diagram depicts how the user will interact with our device. Once the user clicks the start button on the app, a signal is sent to the microcontroller over Bluetooth to initiate the transmission of sensor data and start the video stream. Once the accelerometer detects that the car is now moving forward, the hardware will re-enter its low power state.

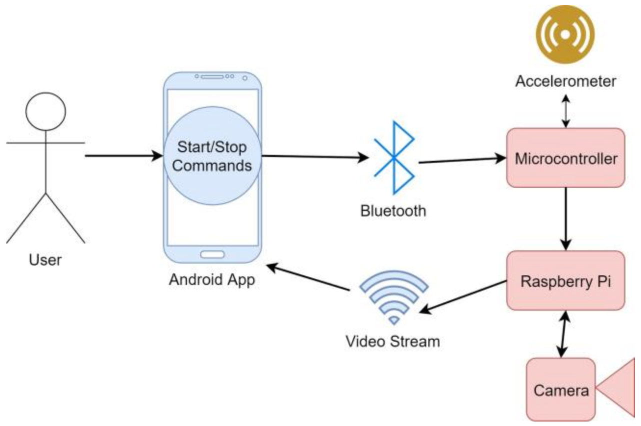

Fig. 2. Use Case Diagram

III. Relevant Technology and Competing Products

A. Competing Products

Aftermarket backup camera solutions are not a new technology, our implementation is simply a different vision that what is currently available to consumers. During our research we came across multiple solutions that allow the user to attach a camera system to the rear of their car. These solutions gave us an insight in what is already available, showing us what works and what needs improving.

The most basic of the competing products that were researched, the ZUS model is simply a backup camera transmitting only video feed, with no other sensor data. The device itself measures only 12.2 X 1.7 inches on the front and weighing 1.42 ounces. This gave us a starting point for what is the basis of an aftermarket backup camera, however its lack of features left a lot to be desired.

We learned from this that the camera doesn't take up a lot of space in the design of a backup camera. When it comes to simply transmitting that feed, there is a lot of room for extra features to be added. While the smaller form factor of this design was nice, we couldn't stuff all of the sensors we wanted into something quite as small, so we knew that was a flaw of this design.

Pearl's RearVision model is closer representation of what our design started out as. It features a bit more complexity, with solar panel rechargeability and a dual camera system for depth perception. However, this product still only provided the user with a video feed and no other data. The form factor of this product more closely resembled the early schematics of what we had in mind for our design. Aside from that there was not a lot this product offered compared with the ZUS model.

FensSens is the only of the researched products that features a sensor system, but surprisingly lacks any camera feed whatsoever. It relayed the sensor data back to a smartphone app, with a graphic that gave the user a representation of any obstructions around the car. This model didn't have rechargeable batteries, but rather ran off AA batteries that would need to be replaced. This approach is a bit of an inconvenience when it comes to user experience, but since the battery life is said to be 5 months, the system can allow it.

Multiple aspects from these systems would be great to have as an aftermarket product for a car, we used this information and found that these systems were lacking features from one another. The form factor of the Pearl allowed for more components to be added, such as the motion sensors in the Fens Smart.

B. Wireless Communication

The three main wireless technologies we were deciding between were Wi-Fi, Bluetooth Low Energy (BLE) and Zig-Bee. Wi-Fi was found to be too power hungry for the applications associated with UART and I^2C, the bandwidth was more than enough. However, for the transmission of video footage, Wi-Fi was the best option. It was selected in conjunction with the Raspberry Pi to encode and transmit the video feed.

For the transmission of data between the phone and the microcontroller, it was a debate between Bluetooth and Zig-Bee. Zig-Bee beat our Bluetooth in most categories, that being bandwidth, range, and even price. The reason we went with Bluetooth in the end however is that Bluetooth is already built into the smartphone. Using Zig-Bee would have required the use of another device to receive data from the microcontroller, making the use of a smartphone redundant.

C. System Enclosure

The enclosure needed to be able to not only house all the components of our system but stay affixed to the car while in motion. Laser cutting, and 3D printing were the 2 solutions that seemed most appropriate, but after further investigation, laser cutting would not work. Laser cutting only allows flat surfaces to be cut, forcing us to create a box shape by hand.

During the final stages of our development, it turned out that the 3D print design we created was too large for the printers on campus. And we tried outsourcing to a third party for printing, but it turns out that was too expensive, so we opted to create the final design by hand, using wood, which gave us a good enclosure for a proof of concept.

The size of our enclosure factored in two things: the size of the components we chose, and the area around the license plate that was available to us. The largest component we have is the microcomputer, which is why the bottom section of the enclosure has its thickness. The dimensions of the enclosure, shown below, measure 165 x 328 x 25mm.

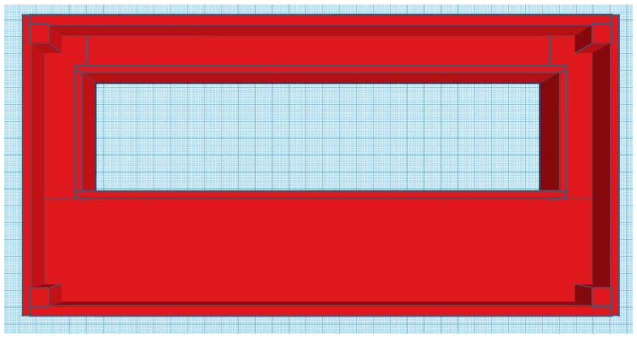

Fig. 3. Backup Buddy 3D Model

IV. STANDARDS AND DESIGN CONSTRAINTS

Our design fit into multiple categories, such as wireless transmission, vehicle and driving safety among others, and thus many standards have already been put into place that we felt our obligation to adhere to.

A. Standards

The only standards found regarding lithium ion batteries referred to their manufacturing. As they can be explosive and dangerous if not handled properly, we had to ensure we were getting them from a reliable source, and that when not in use they were stored properly.

For the C and Java code we were writing, there are certain conventions in the community that allow for code reuse and readability. Both Java and C documentation regarding their respective conventions were read and we made sure to stick to these conventions. The most important of these that we stuck too had to do with writing code that can be run on different systems, as well as how the compiler will handle our code [2] [3].

Android applications that make their way to the Google Play store need to adhere to Googles terms of service. When requesting permissions from the user it is imperative that any and all data that we need access to be made aware to the user. Because of this, the first time the app is downloaded, the user is asked to allow for any permissions for data that might be needed for the app to run properly.

Section VIII goes into detail of our testing process, but there were standards when it came to software testing. While quite an extensive list, we took away the following details.

Anytime our code is updated in a major way, we had to re run our tests to ensure everything was working as our tests expected. The next was documentation, and the various titles of documents to keep track of. During testing we took notes that we exchanged as a group regarding the different tests, but once our testing divulged into the hardware and software testing together, our documentation turned into spreadsheets of what needed to be retested or improved. We also created tests around bugs that came up during our testing, as well as remove tests that passed due to bugs [4].

B. Design Constraints

Our initial research found multiple constraints that might hinder our progress during the development phase. These ranged from economic, environmental, and even social constraints. When development commenced, we found only some of these impacted us in a meaningful way. The political constraint that we had to work around was the fact that some laws inhibit the use of mounts on or around the license plate. For our testing purposes we made sure that the license plate number was visible when mounted, however this wouldn’t be enough for this to be used all over the country, something that was a challenge giving the size of our components.

Manufacturability of our 3D enclosure was also an issue, the lab on our college campus had numerous printers for students, however due to the size of our enclosure, we had to go off site to get the final enclosure printed. This caused us extra time and money but was a necessary expense.

V. COMPONENT SELECTION

Our system is broken up into various sensors, as well as the use of a microcontroller and microcomputer, our final selection was dependent on both cost, component research, and the experience of our individual group members.

A. Microcontroller – MSP430F5969

The microcontroller oversees handling all of the sensor data, as well as some interaction with the microcomputer and camera, and sending this information to the Android app. For this task we decided to go with a TI chip, our group was already familiar with their line of MSP430 microcontrollers and had a simple one we could use during prototyping. In comparison with 2 other choices, the MSP430G2553 and the ATmega328P, the FR5969 had the lowest power consumption and the most amount of GPIO pins. It was the most expensive out of the bunch, but by only a dollar more, so this did not impact our decision.

The microcontrollers were all similar, as they all had UART and I^2C, which were needed for our sensor data. There was also the added benefit of it being a smaller form factor. The G2553 was one every member of our group already owned, but it was so large and had a limited number of GPIO pins it would have made for a much less optimized PCB design.

B. Microcomputer – Raspberry Pi 3B+

The encoding of the camera feed to the phone needed something a lot more powerful than a microcontroller, leading to us using the Raspberry Pi. Again, our group had the most experience with the Raspberry Pi line of microcomputers as opposed to one of its competitors, the Banana Pi. The 3B+ model had a ribbon cable input, built in Wi-Fi, and exposed pins that would allow us to interact with the microcontroller, which is all that we needed.

C. Camera – Omnivision 5647

With safety being the backbone of our design, we required a wide viewing angle from the camera as well as ease of integration with the Raspberry Pi. There is an official camera for the Raspberry Pi, however it only had a standard lens with a viewing angle of 62 degrees. The Omnivision 5647 was just as compatible, having the correct firmware as well as a wide-angle lens attached to it with a field of view of 150 degrees, ideal for a backup camera. The camera is attached to the Raspberry Pi using a ribbon cable, which was ideal for the placement in our enclosure, allowing rigid flexibility for positioning it at the bottom of the enclosure.

D. Ultrasonic Sensor – HC-SR05

Most of the cheap ultrasonic sensors found online operated very similar to one another, with the only differences being the manufacturer or the color. A 4 pin and a 5-pin model were available, with the 5-pin model having an extra feature. Normally, there is a pin to activate the trigger, which sends out a pulse, and a pin to receive the echo that comes back, and the time between them is measured to calculate the distance to the object the pulse bounced off. This requires 2 GPIO pins. The 5-pin model has an extra pin, that when grounded, allows the echo pin to operate as both the trigger and the echo, allowing for the use of a single GPIO pin. We decided to go with this to simplify our PCB design as well as the layout of all of the hardware.

However, when it came time to test these sensors, the datasheets that were found were sparse and limited in their information. While they boast this single pin trigger/echo feature, it cannot be implemented with the sensors we received. The 5-pin model can still operate with the 2-pin trigger/echo setup, but this did cause a modification to be made to our PCB design.

E. Accelerometer – MMA8452Q

For the use of motion sensing and direction, we wanted to go with an accelerometer. Their small size and generosity with data accuracy gave us a lot to work with. The protocol that the accelerometer uses for communication with the microcontroller is I^2C, one line for the data and one line for the clock. We went with a 3-axis model, as we did not need a gyroscope or a compass along with the accelerometer. A lot of other models gave the same information, however they transmitted it over an analog signal. Considering our use of a digital framework, using an accelerometer that gave a digital signal would allow for better integration.

F. Bluetooth Module – HC-06

For the transmission of data between the microcontroller and the app, we needed to use Bluetooth. As it was already integrated with the Android phone we used, a module that could be soldered onto our PCB would allow us to send information over UART to the phone. Our original module was the HM-10, however its advertised range was nowhere near what we were getting during our testing. The HC-06 was our second choice, which had a much-improved range for our purposes.

The transmit and receive pins on the module are connected to the transmit and receive pins on the microcontroller for UART communication. Every time there is new data to be sent over, we break it up from 16-bit values into two 8-bit values and transmit them in an order that the app knows, so it can be properly reconstructed for the user.

VI. Hardware Design

The hardware involved all of the components listed in the previous section, brought together on a single PCB. One of the biggest concerns with the hardware is the power efficiency. Using lithium ion rechargeable batteries means we have to be a lot more conscious of the power draw from the sensors, Raspberry Pi, and microcontroller. This section covers all of that, as well as electronic components that will be used to connect all the major components.

A. Power Solutions

Being mounted on the outside of a car, we had a few different options how powering our system. We first considered tapping into the brake lights, as these would be the closest source of electricity to the license plate. We opted against it in favor of something that wouldn't force the user to run wires through their car. Rechargeable batteries were the best option for this. In terms of charging them, solar panels were considered. However, because of our enclosure design and the placement of the hardware, even if solar panels were to be affixed, they wouldn't get the best sunlight exposure, and thus were deemed an unnecessary addition.

For the rechargeable batteries, we chose lithium ion over Nickle metal hydride. Lithium ion batteries are typically used in cell phones and notebook computers due to being lightweight yet holding lots of energy within a compact package. Another advantage is their low self-discharge when the battery is not in use. Since lithium ion batteries commonly operate at 3.7 V, the voltage is easily and efficiently able to be stepped down without much power loss since the MCU can operate at a voltage of 3.6 V, only a 0.1 V difference. The biggest advantage of lithium ion batteries is their self-discharge rate, which can typically be from 0.5% - 3% of the batteries capacity per day [5].

B. Voltage Regulators

In our design, all the components on the PCB are designed to have low input voltages - typically between 1.8V - 3.6V. However, with the battery connected, we will have a 3.7V - 5V DC input, depending on the battery we choose. In order to operate our microcontroller and attached components, including the camera, we needed to convert the battery into the appropriate lower voltage. A voltage regulator would have been the cheapest option, required only a resistor. However, since the resistor value never changes it only supplies the voltage intended with a constant load impedance - which may not hold true in a real-world environment. While that is a negative aspect of the reliability - there is a positive aspect in that it is impossible for there to be a short circuit - which means the components are safe. This circuit will waste most of the power supplied into the resistor, which means very low efficiency, less than 50%.

In terms of our design, we decided that higher frequency voltage converters would be more beneficial. This is mainly due to the ability to have a smaller inductor and capacitor in the circuit, as well as the fact that the power loss was minimal when currents are as small as they will be in our device.

Based upon the different voltage regulators we researched, we decided to use the TPS63051 for our 3.3 V applications. The biggest factor in deciding was due to it having a significantly higher switching frequency than the TPS64200 family of devices. The difference between the other characteristics - such as efficiency and the voltage input range were negligible. The major advantage the TPS64200 family had was the maximum current being 3 amperes, but for our design this much current would not be necessary, in fact we only estimated needing 9 mA. The maximum currents of 1 ampere in the TPS63051 will be more than enough to supply power to all of the components that require a 3.3-volt VCC.

C. Logic Voltage Level Shifting

Our system has components that operate at two different voltage levels - 3.3 volts and 5 volts. In order for any components that operate at different voltages to communicate with each other, the voltage level of each data pin must be shifted along the connection path in order to not cause damage to components or give unreliable results. In terms of our design, the specific components that operate at different voltage levels but were required to communicate to each other were the HC-SR05 ultrasonic sensors and the MSP430FR59561 microcontroller.

We opted not to use voltage dividers on the PCB, as it would require multiple resistors, for the 3 ultrasonic sensors used, but instead use a logic voltage level translation. The TXB0104 model can handle 100 mA of output current, as opposed to the LFS0204 mode, which gives us a little more safety in terms of extra unanticipated current [6]. To safeguard our design, we decided to use the TXB0104 for our system

D. Raspberry Pi Power

The Raspberry Pi is the component that will draw the largest amount of power in our system. In order to power the Raspberry Pi, we needed a voltage source supplying 5V. Since our lithium ion battery will not supply this voltage, we will need a DC step up converter, also known as a boost converter.

The trade-off when deciding between the TPS61253A and the TPS61232 boost converter components was the switching frequency and the maximum current output. Since this portion of the power supply would be used to power the Raspberry Pi module, which recommends for a 2.5 ampere power supply, we decided to use the TPS61232. This component, despite having a lower switching frequency, still held efficiency above 90% at the low current ranges that our system would operate in low power mode. It also was a safer choice due to being able to handle more load current, in case of any unforeseen current drains in our implementation.

	TPS61253A	TPS61232
Voltage input range	2.3 V - 5.5 V	2.3 V - 5.5 V
Voltage output range	5 V	5V
Maximum current output	1.5A	2.1A
Operating temperature	-40°C - 85°C	-40°C - 85°C
Switching Frequency	3.5MHz	2MHz

Fig. 4. Boost Converter Comparison

E. Initial Design

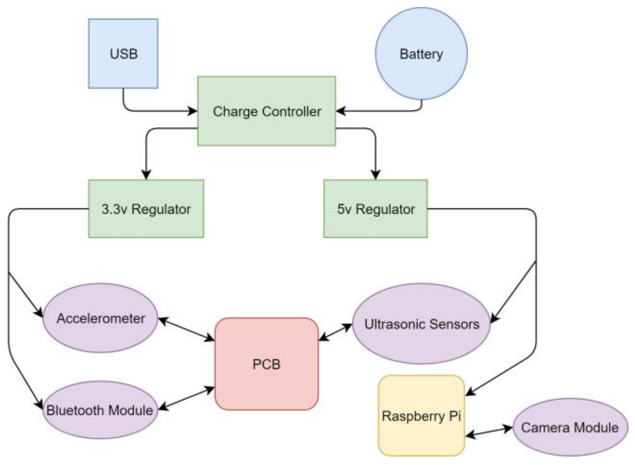

Fig. 5. Initial Design Flowchart

This image serves as the basis of our hardware design. The items in purple represent the sensors and hardware that are supplement to the microcontroller and the microcomputer. The battery, as well as a USB charging port connected to the charge controller allow us to charge the batteries while in the device. The branches to the respective regulators then go off to the hardware that needs either 5v or 3.3v to be powered.

F. Power Design

The step-up converters were used for up stepping the power for the Raspberry Pi, we needed 2 different voltage regulators for the many parts of the system: a 3.3v regulator to power the microcontroller and accelerometer, and a 5v regulator for the ultrasonic sensors and Raspberry Pi. We used the TPS63051 buck converter and the TPS61232 step up converter for our 3.3v and 5v needs respectively. These modules were chosen based on the 3.3v regulators efficiency, while the 5v regulator was designed specifically for lithium ion batteries.

G. Microcontroller Design

Part	Manufacturer	Model	QTY
HM-10 Bluetooth Module	DSD Tech	ML-HM-10	1
MMA8452Q Accelerometer	Xtrinsic	SEN-12756	1
HC-SR05 Ultrasonic Sensor	Iduino	HCSR0501	3
TXB0104 Level Shifter	Texas Instruments	TXB0104DR	1
4.7 µF X5R Capacitors	Taiyo Yuden	EMK212BBJ475MK-T	1
0.1 µF X5R Capacitors	Taiyo Yuden	JMK042BJ104MC-W	8
4.7k Resistor	ROHM Semiconductors	ESR01MZPJ472	3
50k Resistor	Vishay	CRCW040250K0FKED	1

Fig. 6. Bill of Materials for Microcontroller schematic

The HC-06 Bluetooth module communicates with UART for receiving and transmitting data. Those pins on the module connect with the MSP430FR59691 on its respective UART pins 2.5 for receiving data and 2.6 for transmitting data. The other most notable pin on the HC-06 is pin 12, which is where the 3.3-volt power source will be supplied. For implementation, we plan on soldering the Bluetooth module directly on to the PCB.

The MMA8452Q accelerometer connects using both I2C and GPIO connection. The SCL and SDA pins each have a pull up resistor configuration in order to avoid floating values and are connected to the SCL and SDA pins on the MSP430FR59561. The interrupt pins are connected to generic GPIO pins on the microcontroller, and the inputs have decoupling capacitors in order reduce high frequency noise in the power supply - as recommended by the datasheet. These interrupt pins were included in case we decided we could use them, which ended up not being in our final design.

As noted before, the three HC-SR06 ultrasonic sensors were originally designed to work on a 3-pin model, but this proved to be a challenge and we had to include 2 GPIO pins for each sensor. The trigger and echo pins on each sensor are connected to a logic level shifter before connecting to a GPIO pin on the microcontroller.

VII. Software Design

A. Android Application

Java was the primary language used when developing the Android app. We utilized Android Studio development and debugging, due to its support from Google as well as the ease of creating new screens for the user to navigate through. The app works through a series of panes: The wake-up command screen, the video stream screen, settings, and the debug screen which we used during testing. A Bluetooth connection is made from the wake-up screen, and once the connection is successful, the video stream begins to initialize, and the user is brought to the streaming screen. Java handles raw byte data a bit differently than C does, so the data being sent from the microcontroller is put back together and converted back to the original integer and floating-point values. Once the microcontroller sends the signal that the car no longer needs to be sending the camera feed, the app with cease all transmissions and go back to the wake-up screen, awaiting the user to need it again.

We went with a very simplified approach for the app, since it will be used while driving, we didn't want the user to have to navigate through a lot of options and pages. There are multiple threads handling the Bluetooth and video streaming actions in the background, to ensure there is minimal lag during peak data transmission.

B. Microcontroller Embedded Programming

TI has very useful launch pads for a lot of their MSP430 Microcontroller devices. Our model had one available, which gave us a lot of room to test and prototyping our code and hardware before soldering to our PCB. The launch pad also allows us to connect 2 wires to the microcontroller that is connected to our PCB to load our code using Spy-bi Wire.

For writing the C code that would be used on the MSP430, we used both Code Composer Studio, otherwise known as CCS, as well as TI's own IDE, Energia. Code Composer allowed us to modify actual registers and use interrupts and low power mode, giving us the most usage out of the device. Energia, while limited in its functionality, was great as a backup tool that we used when we needed to test thing such as pins going high without creating a whole new project in CCS.

The software for our microcontroller had the responsibility of turning on and capturing data from the accelerometer and the ultrasonic sensors, and relaying that data over UART to the Bluetooth module, being taken in by the Android app. The code that is written to run the sensors and begin collecting data, waits for an interrupt that runs when the wake command is sent from the phone to the microcontroller. During the time that the camera is in use, the sensor data is constantly being sent. It isn't until camera feed transmission ceases that the microcontroller will stop sending data and re-enter its low power state. It is this low power state that allows for better battery consumption. The Raspberry Pi is very power hungry, so maintaining a system that shuts down these large operations when the user is no longer backing up extends the life of the battery.

C. User Stories

Below are the user stories that we created when designing the software for our application and hardware. We wanted to look at the user experience from their point of view, and ensure it functioned as a consumer would expect. These stories were formatted into requirements and aided us in the design process.

ID	As a...	I want to be able to...	So that I can...
1	User	See the camera feed from the rear facing camera	see what is behind me as I backup
2	User	Get visual cues when an obstruction is behind the car	act accordingly so that a collision does not occur
3	User	Get audio cues when an obstruction is behind the car	act accordingly so that a collision does not occur
4	User	Connect to the camera assembly via Bluetooth	have it turn on before I get to the car
5	User	Connect to the camera assembly via Wi-Fi	get the various sensor information sent to my phone

Fig. 7. User Stories

All these user stories involved communication between the app and the hardware. The audio and visual queues start off with the sensor data being transmitted to the app, which begins with the Bluetooth connection being initiated. As a user, we included the stories about Bluetooth and Wi-Fi, since Bluetooth is a much simpler process for end users to configure. The app is designed to seamlessly make the connection, and even if the user had to find the device on their own, they would be able to. This was initially part of our design, having us consider if there was multiple Backup Buddies to connect to.

VIII. System Testing

Testing the android app and the embedded hardware was done as we integrated new features. Once these features were set in stone and everything was brought together, we created a series of physical and tests to put the device through, both in the lab and in a testing environment

A. Ultrasonic Sensor Testing

Due to the ultrasonic sensors not being the most accurate devices, we recorded the data we received from them to create thresholds and maximums to stop incorrect data from being sent to the user. We created a maximum distance to be considered at 130cm, as anything further than this would not be in immediate danger of a moving vehicle. This was tested by having the sensors placed facing outward on a table, and a large flat notebook as the test obstruction. Our testing showed us that anything further than 130cm wouldn't be relayed back to the user.

We also had to test for 0 data. This is what we called that data that resulted from a timeout of the sensors. Because they rely on a pulse to be caught, if that pulse bounced out of range then the data wouldn't be useful. So, we had our system set to not use the data if it was zero, but maintain the value caught before that transmission. These tests proved successful, and we were able to have our sensors give us usable data, even if there were spikes or drops.

B. Accelerometer Testing

The 3-axis accelerometer we are using has a lot of different registers, but we were just interested in the x, y, and z Gs data. This was simple enough, as we just programmed the accelerometer to send this information over Bluetooth alongside the ultrasonic data. The debugging interface we created showed us that this data was coming in as expected and changed with regard to changed in motion of the accelerometer.

C. Camera Testing with Sensor Data

Transmission of the camera feed along side the data incoming from the Bluetooth is what the user will be seeing when they are inside the car. We setup an environment where the app would send a signal to initiate the transmission with both the microcontroller and the microcomputer. This initiated both a Bluetooth and Wi-Fi connection, starting the transmission of both the camera feed as well as the sensor information.

Inside of the lab, we had the camera sitting in safe place, with the sensors on the PCB connected separately. While the transmission of the video feed was going, data was being sent to the phone. As expected, whenever we would place an object in front of the sensors, there was a visual change, depending on how far the object was. We were able to view the raw data on our computers, to verify that the data we were seeing gave the appropriate response on the phone.

ACKNOWLEDGEMENT

The authors wish to acknowledge our professors, Dr. Lei Wei and Dr. Samuel Richie, for their advice and feedback during the design process, as well as all the faculty and industry reviewers who took time to come out and critique the final product.

REFERENCES

[1] United States, Congress, Cong., Committee on Transportation and Infrastructure. "NHTSA." *NHTSA*, Nov. 2008. 110th Congress, 2nd session, report DOT HS 811 44 , crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/811144

[2] "Java Code Conventions" *Sun Microsystems, 1997* *http://web.archive.org/web/20090915035719/http://java.sun.com/docs/codeconv/CodeConventions.pdf*

[3] "ISO/IEC 9899:TC2 Programming Language - C" *ISO/IEC,* 2005, http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1124.pdf

[4] Hesham Alaqail and Shakeel Ahmed "Overview of Software Testing Standard ISO/IEC/IEEE 29119" *IJCSNS International Journal of Computer Science and Network Security,* VOL.18 No.2, February 2018, https://www.researchgate.net/publication/323759544_Overview_of_Software_Testing_Standard_ISOIECIEEE_29119

[5] "What's the Best Battery?" *Battery University*, 21 Mar. 2017, www.batteryuniversity.com/learn/archive/whats_the_best_battery

[6] "TXB0104 4-Bit Bidirectional Voltage-Level Translator With Automatic Direction Sensing and ±15-KV ESD Protection." *Texas Instruments*, Apr. 2006, www.ti.com/lit/ds/symlink/txb0104.pdf.