# IoTECH
# Internet of Things Extensible Car Hub

Nigel S. Paine, CSE; Raghid Bahnam, EE; Nicholas A. Korniyenko, EE; Christopher R. Ingerson, CSE

**"IoTECH", also known as the Internet of Things Extensible Car Hub, is an IoT device that interfaces with the OBD-II diagnostics port and a variety of external sensors. In an era of WiFi-enabled devices, we seek to create an IoT platform that is unique to automotive systems. This platform will allow the user to create an application that will interface with an array of sensors and communicate serially or wirelessly via Bluetooth, WiFi, and 3G data. User data is transmitted via 3G to a web server where notifications are then broadcast. A few sample applications have been implemented with IoTECH such as the "TempAlert" system which notifies a user if there is a person or pet in the car and a certain temperature threshold is reached.**

## I. INTRODUCTION

We see more IoT devices integrated into homes across the country, but there aren't such devices out there for automotive applications. Nest thermostat and Hue Philips Light Bulbs are examples of IoT devices. Both are app controllable where prefered lighting and temperature can be controlled via app. Nest can be programmed to set the temperature when the user comes home where Hue can change each light bulb to a prefered color and brightness [21][22]. Existing automotive IoT devices are limited to the space underneath the steering column where the devices plug into the OBD-II port [13][14][15]. These devices may have a variety of sensors but are limited to being under the steering column. For example, a device is able to give GPS coordinates but not give a picture of a breakin, or detect motion of an endangered child due to temperature. What we plan to create is a "smart hub" which will be directly connected to a vehicle's OBD-II port but is also extended with sensors to produce many more different applications that can't be done under the steering column. This includes infrared sensing and picture taking done by this extension. This IoT device would be both extensible and modular, capable of integrating with most sensors via a hardwired or wireless connection. One of the main goals of this device is to receive live-updated information directly to the user's phone, whether it is getting temperature alerts or notifying the user if the car has been hit. We hope to extend the usability of the OBD-II port beyond its current capabilities using our technology. Our device could save lives of children left by their parents in hot cars, catch thieves red handed, report that your car got hit, and so on forth. It will make this country safer. The applications can be limitless. We want this device like home IoT devices to be extended not only to a few applications and sensors but to maybe hundreds to suit each person's needs. Table 1 shows the hardware specifications for our Hub and Extension. It contains the initial requirements that we set for ourselves and what our actual prototype specifications were. Some of the specifications were accomplished while others would be our next goal to accomplish.

| Specifications | Initial Requirements | Actual Prototype |
|---|---|---|
| Small/Portable | Hub size of a phone (2.5"x2.5"x2.5") w/ extension (5"x2.5"x3") | Hub (6.5"x3.75"x2"), Extension (5"x1.75"x3.25") |
| Lightweight | Be able to drive car without device falling down and damaging diagnostics port (hub <100g & extension <200g) | Hub: 388g & Extension: 260g |
| Extensible | Supports Serial, Wi-Fi, Bluetooth, and 3G communications | Accomplished |
| No lag communication | Notification Delay < 1 min | Notification Delay < 1 min |
| Long-lasting car battery life | Lasts 160-200hrs (~1 week) | Car Battery Life 8 Days, |
| Long-lasting extension battery life | Lasts > 12hrs. | Lasts: 14 Hrs. (Can be recharged) |

*Table 1. IoTECH System Specifications*

## II. DESIGN

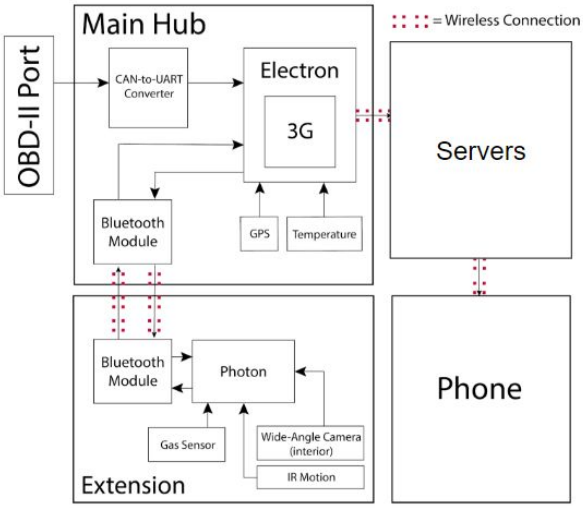



*Fig. 1. IoTECH Signals Block Diagram*

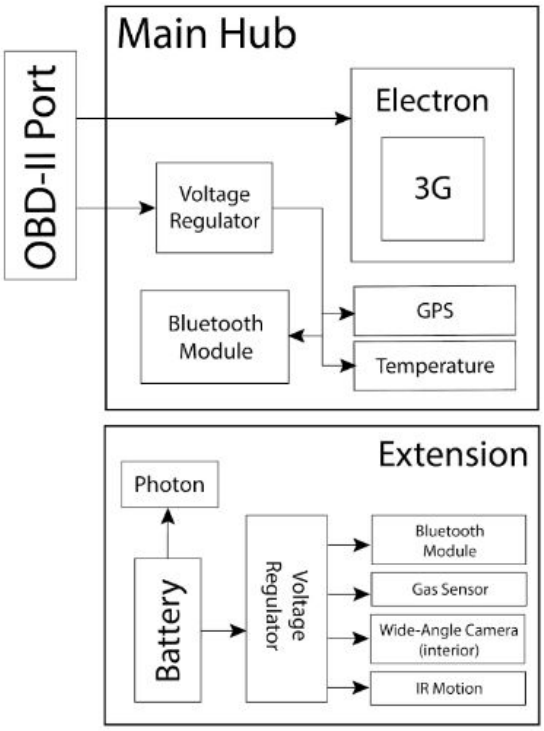



*Fig. 2. IoTECH Power Block Diagram*

### *A. IoTECH Hub*

The most important part of the design in Figure 1 is the main hub. For our hub, we are using the Particle Electron [24] as the main device. The main goal of our hub is to request data from the OBD-II port and extension and use the data from the OBD-II port, extension, and its own external sensors in order to run a set of designed applications.

The OBD-II port provides crucial diagnostic information about the current status of the car, and also provides constant power for the Electron. It supplies 12 volts, which can be fed directly into the $V_{in}$ pin of the Electron. The Electron has a built-in voltage regulator that can take an input of up to 18 volts and converts it to the 5 volts necessary to run the Electron. More information about the power supplied by the OBD-II port will be provided in later sections.

Attached to the Electron, we have a Bluetooth module [25] that is able to accept communications from the IoTECH Extension; the extension will be explained in the next section. The data received by the module is transmitted to the Electron via the TX/RX pins, where the data can be stored in variables and used in applications.

We have two sensors on the main hub, one that measures the temperature and humidity of the ambient environment inside the car, and a GPS [41]. The purpose of the temperature/humidity sensor is to make sure that any young children or animals left in the car, whether intentionally or accidentally are not in any danger as temperatures rise in the summer. All of the variables are processed in the firmware of the Electron, and once an application evaluates to true, an event is published to the cloud via 3G. Communication with the server and the integration of If This Then That (IFTTT) [26] will be discussed later. A sample application may use the on-board GPS which returns the current location of the user's vehicle. This data can be used in parallel with the Google Maps API [40] to return the speed limit of the road a user is traveling on. That speed limit can be compared to the car's current speed, as retrieved by the OBD-II port, and if a user is traveling too fast, similar notifications can be sent through IFTTT. All the circuitry for the Hub was mounted on the PCB shown in Appendix Figure 12.

### *B. IoTECH Extension*

The next biggest component in the IoTECH device, and more of a proof-of-concept than anything is the IoTECH "Extension". The extension's role is to "extend" the functionality of the main smart hub and allow users to interface with external sensors via Bluetooth Low Energy (BLE) connectivity enabled by the HC-05 Bluetooth module [4]. This device is powered by a 3.7V Lithium Polymer (LiPo) battery [17] and holds any sensors that require better location placement in the car (i.e. not under the dashboard). The LiPo battery is boosted to 5V with a step-up voltage regulator [18] to support powering any sensors relying on a 5V input. The extension also has built-in charging capabilities with the charging module that is based on the MCP73833 microchip, that can charge

Li-Po batteries with up to 1000mA of current [38]. This module is seen in Figure 3. Everything is mounted on a PCB that can be seen in Appendix Figure 13 and put into an enclosure seen in Appendix Figure 14.

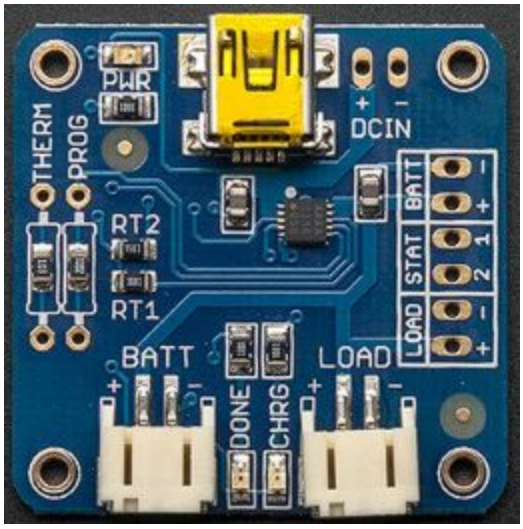



*Fig. 3. USB Li-ion/LiPoly Charger- v.1.2 [38]*

The main processing unit on the extension is the Particle Photon with built-in WiFi capabilities and firmware supporting the Particle API in addition to many of the libraries from the Arduino API [27]. The main programming development environment being used to upload code to the board is the Particle Build Online IDE [28]. Note that previously, the Redbear Duo BLE/WiFi microcontroller [8] and Arduino IDE [19] were being used in our extension. After development with the Redbear Duo, we found it in our best interest to use the Particle Photon instead for a more compact design and better integration with the IoTECH system.

The first main sensor used on the extension is the wide-angle camera. The camera we chose is the LinkSprite JPEG Serial UART Infrared camera. This camera can have a variable resolution but has a default resolution of 160x120 with night vision enabled by infrared LEDs and up to 120 degree field of view [9]. Serial communication is enabled by connecting the camera to the extension multiplexer chip which connects to the Particle Photon serial pins (pin D16 and D17 or the RX/TX pins as seen in Figure 4 with the Particle Photon pinout map). The multiplexer is a new hardware feature in the IoTECH extension since the Particle Photon only has a single set of serial TX/RX pins so we have to use the multiplexer as a "switch" between the serial bluetooth module and the serial camera using a select pin. The multiplexer chip we opted to use is the Texas Instruments SN74HC157N chip [29]. The given Arduino code for the serial camera is not readily compatible with the Particle Photon due to

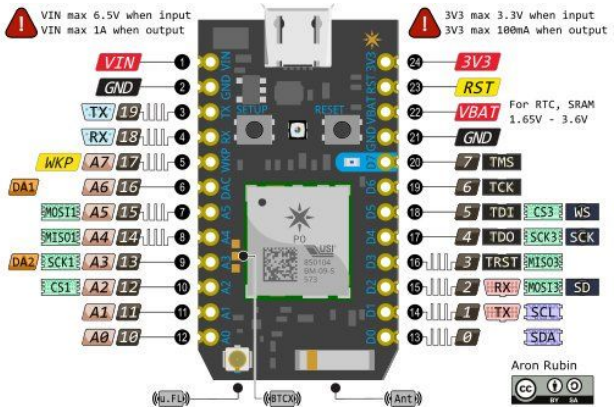



*Fig. 4. Particle Photon Pinout*

hardware limitations (i.e. buffer size) so changes in the code have to be implemented to make sure that when serial data is sent over to the Photon device, the data buffer does not overflow. In addition, the baud rates must also be set accordingly to avoid data transmission errors.

Camera picture data is sent as a stream of hex values formatted in JPEG format. Once the camera data is on the Particle Photon the multiplexor handles the switch between the camera to the bluetooth module. The JPEG image is then transmitted line-by-line to the IoTECH hub, where another serial HC-05 bluetooth module is being used to receive the data and store it on the Particle Electron. The image is about 50KB and takes a little less than a minute to transmit via Bluetooth Low Energy (~30 seconds to take the picture and ~30 seconds to transmit via BLE). The image is then sent from the hub to the back-end server via a 3G mobile connection and takes about 20 seconds (see IoTECH server for more info on implementation). The image is then cleaned up by a backend Python script and converted to a nicely-formatted JPEG image file which is then uploaded to the Cloudinary database [30] and sent to a mobile phone using a custom image URL and the Twilio API [31]. In total, the transmission time for an image from extension to phone is about 2 minutes. To speed up transmission time, in the future, a better camera should be implemented and WiFi should be used for large data transfer (i.e. image/video files); this could significantly cut down on transmission time to potentially transmit a full-size JPEG image in under 30 seconds [32]. Just for comparison, the HC-05 BLE protocol transmits at a *maximum* rate of 375KB/s whereas our image size is 50KB so theoretically we could transfer it in under a second, but of course with interference and distance between the modules that data rate goes down significantly. Also, a limiting factor is the default baud rate of 38400 which we

were unable to test any higher baud rates for, which may have potentially also increased transfer speed [33].

The second sensor being implemented on the extension is the HC-SR501 passive infrared (PIR) motion sensor. This sensor will be able to communicate in parallel with the camera so that when motion is triggered, the camera turns on and snaps a picture of the car's interior. The delay on the motion sensor can be set directly on the device (ranging from 0.3 seconds to 5 minutes) or via software. The motion sensor also has an angle of view of up to 110 degrees and up to 7 meters away [10].

With the motion sensor and camera, the "TempAlert" system can be implemented more robustly, allowing the user to only get notifications when there is motion detected in the car in addition to receiving a low-resolution picture (to avoid high 3G data consumption) identifying who or what is in the car. TempAlert is a proof-of-concept application developed by our team incorporating IoTECH allowing the user to receive notifications if the internal temperature exceeds a set threshold while also detecting motion and capturing an image to ID false positives.

A third sensor implemented on the IoTECH Extension is the MQ2 Gas Sensor [36] which can detect the presence of alcohol, Carbon Monoxide (CO), and smoke. The gas sensor uses a nonlinear but proportional function of gas concentration to voltage, so as the gas concentration increases the voltage also increases. The sensitivity used may change depending on the gas being detected. Figure 5 shows a better depiction of which gases are able to be detected with the MQ2 gas sensor and the range of PPM (Parts Per Million) for a particular gas it can detect in the air surrounding the sensor. For this sensor we created the GasAlert application which simply sends a text to a user whenever alcohol is present in the vehicle. While we chose alcohol the user could setup the sensor to detect something different such as CO. As proof of concept, the alcohol sensor was tested using mouthwash and hand sanitizer, both containing some level of alcohol. The Photon in the extension would then send a trigger via BLE to the Electron in the hub which would update the trigger via 3G in the Particle Cloud [37], allowing a text message to be sent to the user alerting them of the presence of alcohol in the car.

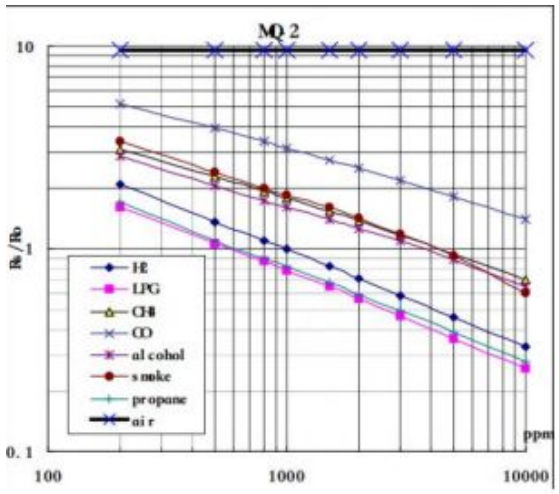
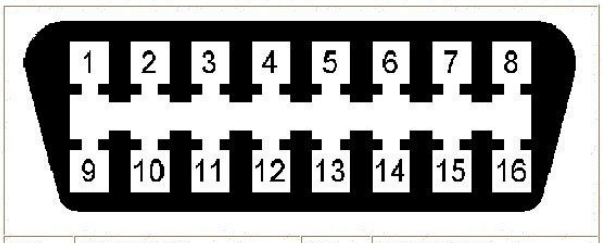



*Fig. 5 MQ2 Gas Concentration for Varying Gases [36]*

All hardware components were tested using the Particle CLI (Command-Line Interface) and a terminal [34] to validate that the correct data is being sent/received. The built-in Particle CLI serial monitor echoes data via the USB-connected Photon. Data tests were also run in parallel with the IoTECH hub on the Particle Electron microcontroller to validate that data was being sent via Bluetooth in a secure and consistent manner.

### *C. OBD-II Communication*




| PIN | DESCRIPTION | PIN | DESCRIPTION |
|---|---|---|---|
| 1 | Vendor Option | 9 | Vendor Option |
| 2 | J1850 Bus + | 10 | j1850 BUS |
| 3 | Vendor Option | 11 | Vendor Option |
| 4 | Chassis Ground | 12 | Vendor Option |
| 5 | Signal Ground | 13 | Vendor Option |
| 6 | CAN (J-2234) High | 14 | CAN (J-2234) Low |
| 7 | ISO 9141-2 K-Line | 15 | ISO 9141-2 Low |
| 8 | Vendor Option | 16 | Battery Power |

*Fig. 6. OBD-II Pinout [6]*

Figure 6 shows a pinout diagram of an OBD-II port. Reading OBD-II was done in few stages. OBD-II uses three protocols to communicates with the outside world. Since 2008, one protocol was mandated in every small size car in the USA. It is known as the CAN protocol [23].

With speed up to 1 mbps, it is the fasted protocol that OBD-II uses. IoTECH works only with cars that uses CAN protocol. Before being able to read any data, the 12V of the car's battery needed to be regulated to 5V and 3.3V to provide the required voltage that the system needs to operate. This was done by using L7805C that regulated the voltage down to 5V. That 5 V was then further regulated down to 3.3V using LE33. This can be seen in Figure 7.

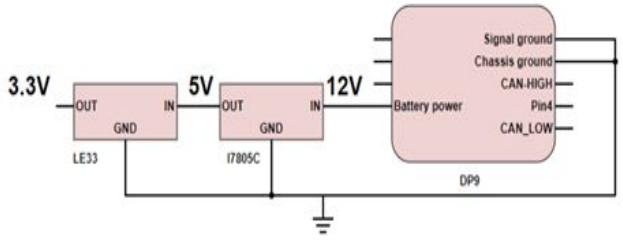



*Fig. 7. Voltage regulator for the car battery*

The system responsible of reading the data consists of three subsystems. They are: a differential subsystem, an interrupter subsystem, and finally the host. CAN bus is a two lines communication. It consists of CAN-H and CAN-L. The first subsystem is a differential chip known as MCP2551. It takes the difference between CAN-H and CAN-L and produce one single. That single is then sent to the second subsystem, The Interrupter. The interrupter is STN1110. It is the world's smallest, lowest cost multiprotocol OBD-II to UART interpreter IC. It has the capability to interpret any protocol to universal asynchronous receiver-transmitter (UART). This is illustrated in Figure 8 below. STN1110 provides an easy means of accessing vehicle data, including diagnostic trouble codes, MIL status, VIN, Inspection and Maintenance (I/M) information, In-use Performance Tracking (IPT), and hundreds of real-time parameters. for the ELM327 command set, while outperforming the original ELM327 IC in every category: stability, performance, and features. Finally, the interrupter is connected to the host through RX, TX ports.

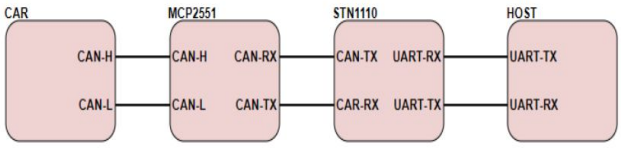



*Fig. 8. OBD II system*

The system is a message-based protocol. The user request specific information for the interrupter by sending a request of HEX values. The Host will then receive a message that has the requested information. OBD-II provides many modes. For now, we are only interested in live data. That is mode 01. Every request starts with 01 to indicate the request for live data. That 01 is then followed by the parameter value. After requesting a specific parameter. OBD-II will respond with another HEX value. That value start with 41 to indicate that it is live. The follow is values that can be translated to real life data using provided equations.

To have a proof of concept, different data was obtained from the car. Speed, RPM, Ambient temperature, and coolant temperature were some of the parameters obtained from the car. Table 2 shows data obtained by requesting specific parameters. The society of Automotive Engineers (SAE) provides equations to translate the data obtained to real live results.

| PID | Tx | Rx | equation | result |
|---|---|---|---|---|
| Ambient Temperature | 0146 | 41 46 3C | A - 40 °C | 68 °F |
| Coolant Temperature | 0105 | 41 05 7E | Value - 40 | 186.8 °F |
| RPM | 010C | 41 0C D1 CC | Value / 4 | 13427 RPM |
| Speed | 010D | 41 0D DE | - | 66 km/h |

*Table 2. Data Requested and Received from OBD-II*

### *D. Server Communication*

Stable connection to a server is a very crucial aspect of the functionality of this project. The hub communicates with the Particle Cloud server using UDP (User Datagram Protocol), which saves a lot of data, as well as power. By not constantly requiring handshakes in order to keep the connection between the hub and cloud active, users are able to conserve their data and make sure that they have available data for when they need it. Acknowledgements are manually required when meaningful data is pushed, however, to ensure that the triggers are received.

When an event is published, a trigger is executed by IFTTT. If This Then That is a free service that can be integrated with Particle systems. In the case of our "TempAlert" application, the service will execute when the necessary prerequisites are met; the temperature is too hot, and movement is detected in the car. When our trigger is executed, users will receive a text message notifying them that their pet or child is in danger.

The backend server handling large data transfer (i.e. camera/video data) is implemented through a virtual Linux machine with the Google Cloud service [38] and is a Java-based server. The server is able to handle multiple concurrent threads and bind these threads to a common IP/port. The main data processing happens with Python through a built-in script that the server calls when receiving a large data file such as an image file. Currently with the TempAlert system, when an image is sent to the

server via 3G, it first has to wait to receive the entire image and store that in a .txt file. The Python script then parses through the .txt file and creates a JPEG file which is uploaded to the Cloudinary server. Then using the custom Cloudinary image URL the Python script is able to send an MMS (Multimedia Message) to a specified user's phone using the Twilio API, as long as the user's phone is validated in the Twilio backend. Both MMS and SMS sample messages are provided in Figure 9.

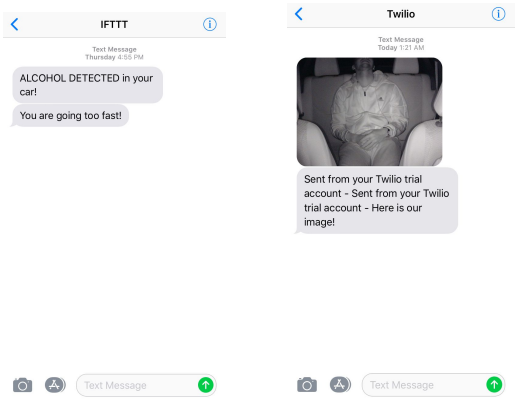



*Fig. 9. Sample SMS IFTTT Trigger (left) and MMS Twilio Image Upload*

### *E. Power Distribution Systems*

Our power distribution system is distributed into two different systems as specified earlier; the Hub and the Extension. These two systems can be seen in the block diagram in Figure 2. Each of these systems run off a different power supply and are vital in running our system.

The Hub is where a lot of our main electronics are located that have sensors on them as well as the microprocessor that sends data via 3G. Its power source is the pin 16 located on the OBD-II port, this pin is what powers the OBD-II devices used to get codes needed for diagnostics. Pin 4 is the chassis ground and pin 5 is the signal ground. This can be seen in Figure 6. This gets its power from the battery of the car which gets recharged when the car is on using an alternator. The voltage on this pin can change and be different at different times. This matters on how the alternator charges the battery and how much the battery can hold charge for [6].

A full software overview of the system can be found in Appendix B. A high-level representation of communication between IoTECH hub and extension as well as all server communication can be seen there as well.

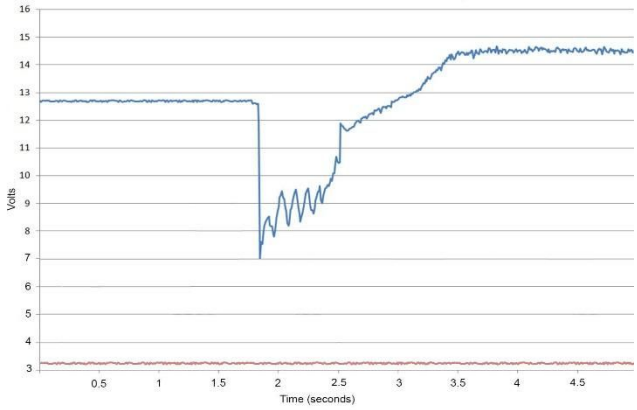



*Fig. 10. Experimental Data for Car Battery Voltage Over Time*

A vehicles voltage also tends to spike in a negative direction and then towards the positive direction when the car is idle and then turned on. Figure 10 shows this very well; the pin starts a little less than 13 volts, as the car gets ignited it drops down to about 7 volts but then increases to 14.7 volts. This of course may be different for different vehicles. Another factor to consider is the temperature. As the temperature becomes smaller the battery's voltage falls 0.01 volts every 10 degrees Fahrenheit [5]. All of this are factors considered in the design hardware circuitry.

The Hub consists of three switching voltage regulators. One located on the hub that is used on the electron microcontroller that has a 3v3 pin that can output 800mA. The model name for it is called the TPS62291 which outputs 1 Amp, can take in 2.3-6 volts, output 3.3 volts, and is 96% efficient [1]. It is used as a pull up for the temperature sensor. The two other voltage regulators that are being used are a 5V and another 3.3V regulator. The 5V is used for some of the OBD-II circuitry as well as the Bluetooth HC-05 module, and the 3.3 V is used for the Particle Electron microcontroller, temperature/humidity sensor SHT10, GPS, as well as some of the OBD-II circuitry [25]. The D24v22F3 is the 3.3 V output voltage regulator that can output 2.6 Amps and take in from 4V to 36V. It has an efficiency of 85-95% [3]. The 5V regulator D24V22F5 is a 2.5A step down voltage regulator that is ideal for the Hub which has a typical efficiency of 85-95% [39]. The reason why we use another 3.3 voltage regulator is because it is fairly economical to buy the 2.6 A regulator and if we wanted to expand our sensor count for other applications we would have the voltage regulator that can handle it.

The Electron microcontroller can take a voltage of anywhere from 3.9 to 17 volts through its power management chip BQ24195 [7]. In the design for the IoTECH we use the Vin pin as a way to power our

microcontroller. It also states in its specifications that a 470 uF capacitor is required in the Vin pin just in case if the electron needs to pull more power when sending 3g data so a 50V capacitor is used [2]. It is directly connected to pin 16 from the OBD-II port since it can handle 11-15 volts. There also is a 470uF 50V capacitor placed at the power source as a surge protection capacitor, just as a precaution of a 40 Volts surge that can happen on the vehicle's battery although this is unlikely. There is also a 50 volt capacitor located on in Vin pin of the 3.3V buck converter which will protect from any surges as well.

The OBD-II circuitry in total takes up to 370 mW of power and only requires 73 mA of current to use it, this is measured value we found. It uses a 3.3V and 5V power supply and as stated earlier will be supplied by the 2 switching voltage regulators that are going to be on the Hub.

Table 3 shows the Hub typical power consumption. This is used to power the application for this system. It shows how long the car battery would be able to pull our system. It is broken up into different components and then computes total power. All these numbers are found from datasheets of all the sensors and computations. We found that the hub would last us about 151 hours of battery life typically and can take up to 3.58 W of power consumption.

| Typical Hub Power Consumption | | | |
|---|---|---|---|
| Components | Voltage (V) | Current (mA) | Power (W) |
| GPS | 3.3 | 20 | 6.60E-02 |
| Bluetooth (HC05) | 5 | 8 | 4.00E-02 |
| Temperature/Humidity (STH10) | 5 | 1 | 6.72E-05 |
| OBD-2 Circuitry (STN1110, MCP2551, 16MHz Clock) | 3 and 5 | 73 | 0.37 |
| Electron (microp) 25 degrees C | 12 | 250 | 3.00 |
| Buck Converter 5V (88% Efficient) | 12 | n/a | 0.05 |
| Buck Converter 3.3V (72% Efficient) | 12 | 0.72 | 0.061 |
| Total | n/a | n/a | 3.58 |
| Average Hours Of Battery Life | 151 Hours | | |

*Table 3. Hub Power Consumption [2][3][4][12][25][39]*

Our extension consists of a Particle Photon as well as one boost converter, a camera and IR motion sensor, gas sensor, and a power management unit. It is powered by a 4400mAh 3.7 volt li-po battery. It consists of 2 electrolytic capacitors. The 16V 100uF capacitor protects from surges to the Photon microprocessor. While the 25V 47uF capacitor is used which is close to the recommended amount for the 5V boost converter. This is recommended because of LC voltage spikes protection [11].

The photon works on a voltage between 3.6-5.5V therefore it is connected directly to the battery [27], while the camera, IR Motion Sensors, and gas sensor work on 5 volts. This is why we use the Pololu 5V Step-Up Voltage Regulator U3V12F5 which outputs up to 1.4 Amps. Although our sensors and bluetooth only use up to 2.175 Watts, which is 435 mA, this is for the purpose of adding more sensors to the extension for other applications [11].

Looking at Table 4 we have the extension temperature alert systems power consumption. It consists of the boost converter, sensors, Particle Photon and the total power and, current [36][9][10][11][27]. It then goes down and breaks up the battery life based on the 4400mAH battery. At first our extension was going to use a 2000mAH battery but then we decided to use a larger battery because it added hours to the battery usage. We found that if everything would have been working at a maximum with high clock frequencies then the battery would last 6.3 hours (2.6W). For typical use 13.5 hours (1.2W) and at a minimum power consumption of 26.6 hours (0.6W).

| Extension Power Consumption | | | |
|---|---|---|---|
| Component | Max (mW) | Typical (mW) | Min. (mW) |
| Boost Converter | 280 | 105 | 60 |
| Sensors | 2050 | 950 | 400.5 |
| Bluetooth (HC-05) (Always On) | 125 | 40 | 40 |
| Photon (Always On) | 148 | 111 | 111 |
| Total Power | 2603 | 1206 | 611.5 |
| Total Current (3.7V) | 704 | 326 | 165.13 |
| Hours Battery Would Last (4400mAH) | 6.25 Hours | 13.5 Hours | 26.6 Hours |

*Table 4. Extension Power Consumption [36][9][10][11][27]*

## III. PROJECT MANAGEMENT

Our team was able to successfully complete all deliverables in addition to delivering some extra functionality for the final demo. All circuits were printed onto a custom circuit board (PCB) and all implemented sensors were successfully working. We were also able to successfully read OBD-II CAN bus data and send/receive commands. In addition, our group was able to implement wireless functionality with the IFTTT server, 3D print enclosures for our IoTECH hub and extension, add in a working camera module, gas sensor and GPS module, create a backend Java/Python server, and integrate that with the Twilio and Cloudinary APIs.

Our two main software developers were Chris Ingerson (CSE) and Nigel Paine (CSE), working on implementing all of the backend code necessary to communicate between devices (sensors, hub/extension, server, etc.). Their expertise also revolved around data processing and interpreting sensor data in a way that is useful for the user. Our two main hardware engineers were Nicholas Korniyenko (EE) and Raghid Bahnam (EE). Their expertise involved circuitry, power management, 3D printing, PCB design, and were the main points of contact for any hardware-related issues.

Breaking down the roles further, Raghid's main area of expertise was OBD-II communication and IoTECH hub PCB/enclosure/power management design, whereas Nick's

area of expertise was power distribution and IoTECH extension PCB/enclosure/power management design. Chris and Nigel worked in parallel to successfully make a working system via software. Each of the team members worked together to fill in where necessary and the workload was pretty evenly distributed throughout with weekly advisor meetings and at least two meetings a week with the team. Time management was essential and Gantt charts with deliverable plans were a crucial component to staying on track.

## IV. CONCLUSION

In a conclusion, we had a successful final demo with several working applications. Several optimization issues have yet to be resolved such as having the server run without resetting and transfer large data such as images at a faster rate (i.e. replacing BLE with WiFi and replacing the old camera module with a more robust up-to-date module). Reliability and security are always a concern so more effort would have to be placed into these two areas. In the future, the IoTECH platform may be marketed to hobbyists allowing them to "hack" their car or to consumers providing them with specific applications such as TempAlert.

## APPENDIX

### A. COST ANALYSIS

| Part | Development | Production |
|---|---|---|
| Circuit Components(Resistors, Capacitors, Voltage Regulators) | $31.40 | $18.74 |
| Gas Sensor (MQ2) | $6.90 | $5.52 |
| Weatherproof Humidity/Temp Sensor (SHT10) | $13.44 | $6.60 |
| PIR Motion Sensor (HC-SR501 ) | $1.17 | $0.91 |
| Infrared Camera (LinkSprite JPEG TTL Serial) | $49.95 | $49.95 |
| Bluetooth Low Energy Module (HC-05) | $3.04 | $2.99 |
| GPS Module (Ultimate Breakout) | $39.50 | $31.96 |
| OBD-II CAN-to-UART Converter (STN1110, MCP2551, DB9 Connector, 16MHz Clock) | $18.60 | $15.40 |
| Particle Microcontrollers (Electron, Photon) | $88.00 | $68.00 |
| 3D Printed Enclosures | $4.55 | $3.54 |
| Printed Circuit Boards (Extension/Hub) | $23.70 | $0.99 |
| **Total** | **$280.25** | **$204.60** |

*Table 5. IoTECH Platform Cost Overview*

In table 5 above we show how much it would cost to develop one an IoTECH device and how much it would cost to produce 1000 of them. As seen in the table, development cost is about 76 dollars more for an individual device purchase versus a bulk purchase.

### B. SOFTWARE OVERVIEW

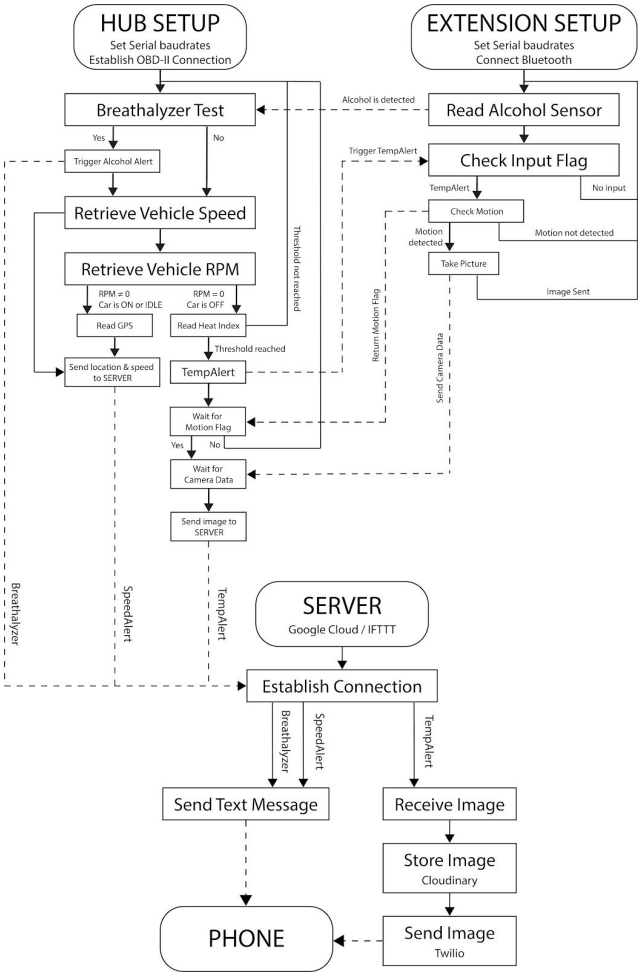



*Fig. 11. IoTECH Software Overview*

### C. PCB DESIGN

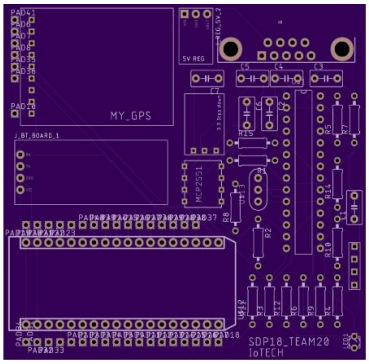



*Fig. 12. Hub PCB*

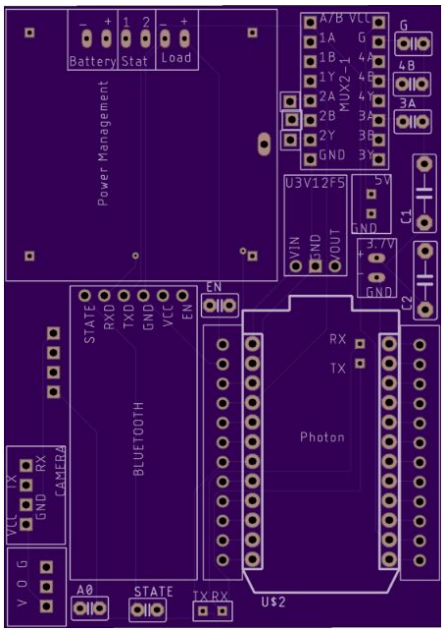



*Fig. 13. Extension PCB*

## D. 3D PRINT DESIGN

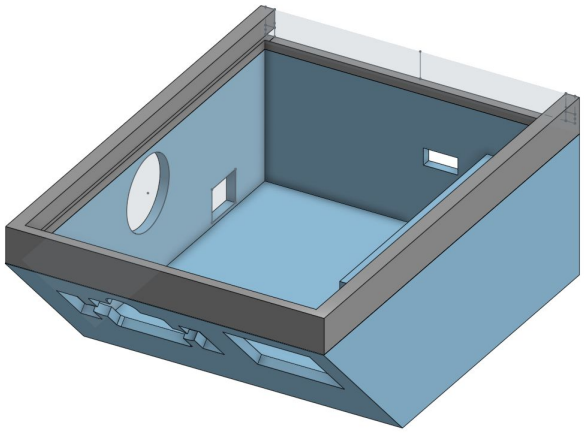


*Fig. 14. Extension Enclosure*

## E. ACKNOWLEDGEMENTS

We would like to acknowledge our advisor Professor Jay Taneja for helping us work through planning our project, from inception to final product. Also a thank you to our evaluators Professor Baird Soules and Professor Michael Zink for helping us improve our device and offering advice on presentation and implementation.

## REFERENCES

[1] Texas Instruments, "TPS6229x 1-A Step Down Converter in 2 x 2 DRV Package ," TPS62290 datasheet, June 2007 [Revised – January 2016].

[2] "Particle," Particle Datasheets Documentation | Electron datasheet.[Online].Available: https://docs.particle.io/datasheets/electron-(cellular)/electron-datasheet/#schematic. [Accessed: 20-Dec-2017].

[3] "Pololu 3.3V, 2.6A Step-Down Voltage Regulator D24V22F3," *Pololu Robotics & Electronics*. [Online].Available: https://www.pololu.com/product/2857. [Accessed: 20-Dec-2017].

[4] Components101.com. (2018). HC-05 Bluetooth Module Pinout, Specifications, Default Settings, Replacements & Datasheet. [online] Available at: https://components101.com/wireless/hc-05-bluetooth-module [Accessed 7 May 2018].

[5] AA1Car, "Diagnosing A Car Battery That Runs Down," *http://www.aa1car.com/*. [Online]. Available: http://www.aa1car.com/library/battery_runs_down.htm.[Accessed: 20-Dec-2017].

[6] "OBD II power when key not in ignition," *Stack Exchange*. [Online]. Available: https://mechanics.stackexchange.com/questions/23047/obd-ii-power-when-key-not-in-ignition. [Accessed: 20-Dec-2017].

[7] "BQ24195 (ACTIVE) I2C Controlled 2.5A/4.5A Single Cell Charger with 5.1V 1.3A/2.1A Synchronous Boost Operation | TI.com," *Texas Instruments*. [Online]. Available: http://www.ti.com/product/BQ24195. [Accessed: 20-Dec-2017].

[8] Redbear, "redbear/Duo," GitHub, 11-Sep-2017. [Online]. Available: https://github.com/redbear/Duo. [Accessed: 20-Dec-2017].

[9] "LinkSprite JPEG Color Camera Serial UART Interface With Infrared," LinkSprite, Nov-2010. [Online]. Available: http://www.linksprite.com/upload/file/1291522825.pdf. [Accessed: 20-Dec-2017].

[10] "HC-SR501 PIR MOTION DETECTOR," Particle. [Online]. Available: https://docs.particle.io/assets/datasheets/electronsensorkit/HC-SR501.pdf. [Accessed: 20-Dec-2017].

[11] "Pololu 5V Step-Up Voltage Regulator U3V12F5," *Pololu Robotics & Electronics*. [Online]. Available: https://www.pololu.com/product/2115. [Accessed: 20-Dec-2017].

[12]Adafruit Industries, "USB LiIon/LiPoly charger," *Adafruit*. [Online]. Available: https://www.adafruit.com/product/259. [Accessed: 14-May-2018].

[13]" T‑Mobile SyncUP DRIVE™ ," *T-Mobile*. [Online]. Available: https://explore.t-mobile.com/t-mobile-sync-up-drive.[Accessed: 20-Dec-2017].

[14]"Snapshot From Progressive | Big Discounts For Good Drivers," *Progressive*.[Online].Available: https://www.progressive.com/auto/discounts/snapshot/.[Accessed: 20-Dec-2017].

[15] https://www.automatic.com/pro/

[16] OBD Solutions "Multiprotocol OBD to UART Interpreter Datasheet'' STN1110 datasheet Oct. 2010 [Revised July. 2012].

[17] Industries, A. (2018). Lithium Ion Polymer Battery - 3.7v 2500mAh. [online] Adafruit.com. Available at: https://www.adafruit.com/product/328 [Accessed 6 Feb. 2018].

[18] Pololu.com. (2018). Pololu 5V Step-Up Voltage Regulator U3V12F5. [online] Available at: https://www.pololu.com/product/2115 [Accessed 6 Feb. 2018].

[19] Arduino.cc. (2018). Arduino Reference. [online] Available at: https://www.arduino.cc/reference/en/ [Accessed 6 Feb. 2018].

[20] Arduino.cc. (2018). Arduino Reference - Serial. [online] Available at: https://www.arduino.cc/reference/en/language/functions/communication/serial/ [Accessed 6 Feb. 2018].

[21] "Hue White and color ambiance Single bulb BR30," *Philips*. [Online]. Available: https://www2.meethue.com/en-us/p/hue-white-and-color-ambiance-single-bulb-br30/46677468941. [Accessed: 12-Feb-2018].

[22] “Your life with Nest,” *Nest*. [Online]. Available: https://nest.com/thermostats/nest-learning-thermostat/overview/. [Accessed: 12-Feb-2018].
[23]EngineersGarage, “CAN Protocol - Understanding the Controller Area Network Protocol,” *EngineersGarage*, 21-Mar-2017. [Online]. Available: https://www.engineersgarage.com/article/what-is-controller-area-network. [Accessed: 12-Feb-2018].
[24] Particle (2018) Particle Guides. [online] Available at: https://docs.particle.io/guide/getting-started/intro/core/. [Accessed 17 October 2017].
[25] HC Serial Bluetooth Products. Available at: https://cdn.makezine.com/uploads/2014/03/hc_hc-05-user-instructions-bluetooth.pdf. [Accessed 16 January 2018].
[26] If This Then That (2018). IFTTT. Available at: https://ifttt.com/. [Accessed 12 October 2017].
[27] Docs.particle.io. (2018). Particle. [online] Available at: https://docs.particle.io/datasheets/photon-(wifi)/photon-datasheet/ [Accessed 7 May 2018].
[28] Docs.particle.io. (2018). Particle. [online] Available at: https://docs.particle.io/guide/getting-started/build/photon/ [Accessed 7 May 2018].
[29] Ti.com. (2018). [online] Available at: http://www.ti.com/lit/ds/scls113d/scls113d.pdf [Accessed 9 May 2018].
[30] Cloudinary.com. (2018). Django SDK–Python file upload, image and video manipulation | Cloudinary. [online] Available at: https://cloudinary.com/documentation/django_integration [Accessed 9 May 2018].
[31] Twilio.com. (2018). Twilio Python Quickstarts for SMS, Voice and More. [online] Available at: https://www.twilio.com/docs/quickstart/python [Accessed 9 May 2018].
[32] Digikey.com. (2018). Comparing Low-Power Wireless Technologies (Part 1) | DigiKey. [online] Available at: https://www.digikey.com/en/articles/techzone/2017/oct/comparing-low-power-wireless-technologies [Accessed 9 May 2018].
[33] Robotshop.com. (2018). [online] Available at: https://www.robotshop.com/media/files/pdf/rb-ite-12-bluetooth_hc05.pdf [Accessed 9 May 2018].
[34] Docs.particle.io. (2018). Particle. [online] Available at: https://docs.particle.io/guide/tools-and-features/cli/photon/ [Accessed 9 May 2018].
[35] Google Cloud. (2018). Google Cloud Platform Documentation | Documentation | Google Cloud. [online] Available at: https://cloud.google.com/docs/ [Accessed 9 May 2018].
[36] Docs.particle.io. (2018). [online] Available at: https://docs.particle.io/assets/datasheets/electronsensorkit/MQ-2.pdf [Accessed 9 May 2018].
[37] Particle. (2018). Particle | Device Cloud. [online] Available at: https://www.particle.io/products/software/device-cloud/ [Accessed 9 May 2018].
[38] Adafruit Industries, “USB LiIon/LiPoly charger,” *Adafruit*. [Online]. Available: https://www.adafruit.com/product/259. [Accessed: 14-May-2018].
[39] “Pololu 5V, 2.5A Step-Down Voltage Regulator D24V22F5,” *Pololu Robotics & Electronics*. [Online]. Available: https://www.pololu.com/product/2858. [Accessed: 15-May-2018].
[40] Google Cloud. (2018). Google Maps Platform - Geo-location APIs | Google Maps Platform | Google Cloud. [online] Available at: https://cloud.google.com/maps-platform/ [Accessed 15 May 2018].
[41] Industries, A. (2018). Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates. [online] Adafruit.com. Available at: https://www.adafruit.com/product/746 [Accessed 15 May 2018].