Willy – Internet Connected Coffee Machine Project Report

Paweł Belczak, Garrett Shirley

May 16, 2019

Contents

1 Introduction

In recent years the world has seen a virtual explosion in the internet connected devices, termed *The Internet of Things* or *IoT* for short. The aim of these devices tends to be simplifying or augmenting an existing device, or technology, by connecting it to the internet or giving it the ability to be remotely and digitally controlled. There are countless examples of these devices, with varying degrees of success. One of the problems with these devices however is that many solve a problem that people don't have and don't need a solution to. From the very beginning the goal of this project was to identify a clear problem people have based off of personal experience and to develop a solution for it.

Traditional coffee machines include rudimentary systems for automatic brewing each day. Unfortunately these tend to be poorly designed afterthoughts that most people write off completely and just deal with having to wake up then make their coffee. This project is designed to eliminate that problem by having the automatic brewing function be the key focus, and to implement it in a way that separates it from the coffee machine itself. This is done by using a smart device, in this case a Particle Photon, that uses a WiFi connection to pull data from the users Google Calender and to brew coffee based off of that data. By separating these responsibilities (brewing coffee, scheduling brewing) into two distinct systems, each system is able to perform better and more user friendly than the more common approach of one device that does everything.

This project and the work associated with it was generally divided into the two distinct areas of Hardware and Software, and the structure of this report and sections within will reflect that distinction. In general, the hardware was designed and implemented by Garrett Shirley, and the software by Paweł Belczak.

2 Project description

The goal of this project is to create a coffee machine that can automate the brewing of coffee by integrating a Particle Photon with an existing machine as well as several backend services. The integration of these components will give the user an easier and more efficient system of automating the brewing of their morning cup of coffee by automatically brewing the coffee one hour before the user's first day event, based on the user's Google Calendar. It will also allow the user to start brewing the coffee remotely through the internet. The coffee machine will also check various conditions in order to ensure safety of the user.

3 Requirements analysis

The main requirements identified for this project are as follows:

1. The machine must be able to brew coffee
2. The Photon must be able to turn the coffee machine on and off
3. The time of the first event in the user's Google Calender must be known by the Photon

4. The machine must still function if the Photon does not have an internet connection
5. The machine must have built in safety checks to prevent damage due to operator negligence
6. The machine must indicate in some way to the user what current state it is in

The purpose of the first three requirements relates to the automation aspect of the machine. The first event of the day must be known as that it is considered to be the time at which user is expected to be out of home, e.g. at work or at school. Given that time the coffee machine can essentially wake the user up and serve him a coffee approximately one hour before that event.

Requirement №4 was included because one of the largest complaints against IoT devices is their lack of even basic functionality without an internet connection, or the worry that discontinuation of support for the device means that the device itself will no longer function[1]. There must be a way to use the device without an internet connection, as above all other details it must be able to brew coffee.

The last two requirements were created to better the user experience, and to help ensure that the machine is used, and functions, as designed. This is done to protect against negligent or malicious use.

4 System design

The two key components at the heart of the design are an existing coffee machine and a Particle Photon board. An existing machine was used because it was readily and cheaply available, and the goal of this project wasn't to make a coffee maker, it was to augment a coffee maker to allow easier automation. The coffee machine chosen is as simple as one can be, it has only a single switch to turn the machine on an off. That was ideal for this project as there is no additional circuitry that must be bypassed or dealt with, the machine can only be either on or off. A Photon board was chosen as that is what was supplied at the beginning of the semester. Any microcontroller that is able to connect to WiFi and has easily accessible digital and analog inputs and digital outputs could have been used instead.

In general, the following design points are described in greater detail in section 5.

The existing machine will be controlled by the Photon via a relay as that is the method used nearly 100% of the time when a significantly large voltage, such as what a standard wall outlet supplies, needs to be controlled by another device. Some sort of physical input on the machine itself will be needed to control it without an internet connection, a momentary switch will suffice. The two greatest safety concerns is the machine being used without water as that may result in large amounts of heat being built up, and the machine being used with water but without the coffee pot in place as that may result in coffee being dispensed freely from the machine into whatever environment the machine is

[1]`http://nymag.com/intelligencer/2016/05/my-internet-connected-home-gadget-hell.html`

in. The first of these concerns will be dealt with by use of some sort of liquid level sensor allowing the Photon to know if the machine's reservoir has water. The second of these concerns requires some way to know the presence of the coffee pot. This is done using a weight sensor installed on the machine, giving the Photon some threshold that the weight sensor must pass in order for the Photon to consider the coffee pot in place.

For the software we will use the FreeRTOS, an operating system which drives the Particle Photon. The Photon will monitor the state of the machine by reading sensors in the main program loop. By monitoring the sensors the software will ensure that the specific conditions are met in order to start the process of boiling the water and the user won't be able to bypass those safety features. Even without the internet connection Photon's software will enable the user to use the machine, while maintaining the safety checks previously mentioned.

The internet functionalities will be implemented by using Particle's Webhooks, Functions and Publish/Subscribe topics, to communicate with the Google Calendar API and IFTTT services. In order to use some of these services the Photon will need to store several authentication tokens. Because a repetitive authentication process can be tedious for the user, the Photon will store required credentials in its emulated EEPROM memory so that they will remain even after cutting off power.

The Photon will once a day, at night, fetch data from the users Google Calendar about the first event in the day, and it will schedule a timer to start brewing coffee approximately one hour before that event. We assume a typical situation where the user wakes up around an hour before his occupation starts, such as work or school, so our coffee machine can make a coffee exactly when needed most, in the morning when the user has just woken up. There will also be a possibility to trigger brewing the coffee through the internet at any time. As previously stated the machine will still be able to be used without an internet connection through use of the manual override button.

5 Implementation

5.1 Hardware

A full schematic and parts list of all hardware used in this project is available in sections 8.1 and 8.2 respectively. This section describes the selection and installment of the hardware used for the various inputs and outputs to and from the Photon.

Bridging the gap between the hardware and software aspects of this project was simple as it was just assigning the various inputs and outputs to specific pins, and describing what each input or output needed to receive from the Photon in order to work.

The Photon itself is powered by a simple USB power adapter like one would use to charge a smart phone. This was chosen due to the fact that the Photon needs a 5 volt power supply to function, and USB adapters are readily available, and have a proven record of safety and reliability. The other option that was briefly considered for powering the Photon was adding to the machine's inner circuitry an AC-DC rectifier, but that would be needlessly complicated and unsafe and was very quickly abandoned. Given a greater amount of time, or in a more refined version, this would be integrated into the machines circuitry as there is no real need for the machine to require two separate power leads.

Because a standard wall power supply was used, there was no real need to consider power usage by the Photon as when idle it uses 80mA at a standard 5V as per the documentation, giving an idle power usage of 400mW. [2]. Using a power cost of €0.3126/kWh[3] gives a yearly cost of running the photon on idle for an entire year of €1.08, an insignificant cost per user per year of use. This could even be lowered in a further version of the project by implementing the Photons deep sleep mode during times where the average user will not be making a coffee.

5.1.1 Inputs

The final design for the chosen coffee machine requires 3 inputs: a force sensor to ensure the coffee pot is in the machine, a float switch to ensure water is in the machine, and a push button so the user can still use the machine without the internet connection. The FSR and float switch function as basic safety checks to avoid needlessly brewing water, or turning on the machine without any water which under a worst case scenario may result in a fire. These take the form of a resistive force sensor (FSR), a fully insulated reed switch and magnetic float, and a standard single pole, single throw push button.

The two switches were attached to the photon using a standard pull down resistor, and the FSR was connected in series with a 10 kΩ resistor to create a 5 volt voltage divider. These were then attached and configured to the appropriate pins on the photon, depending on if an analog or digital value was need from the component.

A force sensitive resistor was chosen as it proved to be the simplest, cheapest, and overall most appropriate method for determine if the coffee pot was in the machine. Other sensors that could be used to do this included a load cell, or some sort of optical sensor. However a load cell would have unnecessarily complicated the design, as would have an optical sensor. The machine does not need to know the contents of the coffee pot, nor an accurate weight of the coffee pot. It simply treats the coffee pot as either there or not, and from the design of the machine, there is a mechanism built in that will only allow coffee to be dispensed if the pot is fully in the correct position. The FSR was attached to this mechanism, allowing for the presence of the pot to be known by the photon in an easy and efficient way. The sensor could not be placed below the coffee pot as that is where the heating element is and from the sensors data sheet, would most likely exceed its operating range. More detail on why the FSR must function in this way is described later in this report in section 6.1.

The float switch used was chosen in a method similar to the FSR previously discussed, namely it was affordable, simple, and provided the exact function that was needed in this project. While the specific functionality of the switch is unknown (no data sheet is supplied by the manufacturer), due to the lack of a direct connection between the main body of the switch and float, it is believed to be a normally open reed switch, with the float containing a small magnet. The main other options for sensing the level of a liquid tend to be optical or

[2]`https://docs.particle.io/datasheets/wi-fi/photon-datasheet/`

[3]`https://www.statista.com/statistics/418075/electricity-prices\`
`-for-households-in-denmark/`

capacitive sensors, both of which require hardware to determine the output of the sensor before passing it to the photon and tend to be suited to applications where more precision is needed. Similar to the FSR, the photon does not need to know the specific water level in the machine, just that there is water so the Photon knows when to start or stop brewing.

One of the largest complaints against IoT devices currently is the potential lack of functionality if the device does not have an active internet connection. Despite the robustness of the infrastructure behind the modern internet, outages still occur, and the last thing someone who doesn't have an internet connection wants is an essentially broken coffee machine. Because of this, it was decided early on in the design of this project that a manual brewing switch needed to be installed, allowing the Photon be triggered to brew coffee without an internet connection. For the prototype, a simple push button one might find on a breadboard was chosen as the only functionality needed was a momentary switch. For simplicity this switch was mounted directly to the protoboard, in a commercial implementation this would be mounted to the machine itself.

5.1.2 Outputs

The Photon has 3 outputs in its final design, these are 2 LEDs used for signaling to the user the state of the machine, if it has enough water and if it has a jug in place, and a solid state relay to control the coffee machine itself.

The LEDs are connected directly to the Photon as they do not have a significant current draw, and are used as a simple status display. One LED corresponds to the coffee pot being in place, the other is used to indicate the presence of water in the machine. When both are lit the machine can begin to brew coffee.

Early during the design process it became clear that a relay was need as there is no other way for the Photon to control the high voltage and high current power the coffee machine needs to function. During the prototyping phase, a standard electro-mechanical relay was used, but this proved to be an endless source of trouble, in addition to needing additional hardware between the relay and the photon due to the power requirements of the relay. To solve these problems, the final version of the project uses a high power solid state relay. This is done mainly to avoid having to add additional hardware and additional points of failure between the relay and the Photon. In a commercial implementation, this would most likely revert back to a traditional relay, as there would be greater resources and time that could be used to perfect the connection between the Photon and the relay.

5.2 Software

The Particle Photon runs on FreeRTOS, an open sourced real-time operating system, widely used in embedded devices, which is then modified and integrated with Particle Cloud. The code is written in (modified) C++ language.

After power on/reset the Photon first (and only once) runs its `Setup()` function. Then it executes an infinite main loop. In between the main loop iterations it checks the connection to the Particle Cloud and executes its Cloud functions, handles topic subscriptions and does other system-related tasks. Our software has to take care of three aspects of our application:

1. Safety features – protecting the user from negligent use.

2. Authentication – handling user token/credentials so as not to bother them with repetitive authentication.
3. Scheduling – fetching data from the API and starting brewing coffee at specific times.

In the following subsections we will take closer look how these three aspects of our solution work together.

5.2.1 Safety features

One of the goals of this project is to make the ordinary coffee machine smarter and more secure to use. We analyzed our particular coffee machine and found two ways how software and hardware can work together to ensure better safety of potential users. This is done by checking if there is enough water in the tank, and checking if the jug is in place. To check these two properties we use two sensors. One analog, that is the weight sensor, and one digital, that is liquid float sensor. Reading values from them is simple and pretty straightforward. The weight sensor gives us values in range 0-4095, where zero means no pressure and 4095 means the maximum pressure. The liquid float sensor gives only two values: 0 when it senses water and 1 otherwise. To start brewing our device needs these condition to be satisfied:

```
if(isBrewing && has_jug() && !noWater) {
    isStarted = true;
}
else if(isBrewing && has_jug() && noWater && isFinishing)
    { isStarted = true; }
```

The function `has_jug()` returns a boolean result of the check if the weight measured by the weight sensor is above or below a given threshold which indicates if the jug is in its place or not. The `noWater` boolean variable inform us if there is water in the tank. The `isBrewing` boolean variable is set by either pressing the physical button or by calling remotely a `Particle.Function()`. The button press is programmed as an Interrupt Service Routine – ISR for short – attached to the Photon's digital pin connected to the button. The interrupt is configured to execute on a RISING signal, so the button press is registered only once if the button is actually pressed. Effectively this means that when the button is pressed, after the current instruction, the Photon's processor will immediately jump to the instruction specified in the ISR, and therefore halt the execution of other code (but only for a very short period of time).

Because the liquid sensor stops sensing the water in the tank at a certain level our coffee machine would stop brewing coffee but would still have water in the tank for about two cups. We noticed that and came up with the solution. When the coffee machine `isStarted` and the liquid sensor starts measuring no water, we set up `isFinishing` variable to `true` so the second part of the if statement resolves to `true`. We also set a software timer for 2 minutes, which is a time, that we measured manually, after which we can be sure that all the water was heated and pumped into the jug.

Because of the conditional statements our coffee machine can only brew when:

- The button was pressed, it has enough water and there is a jug in place, or
- The button was pressed, it has a jug in place, water is not sensed by the liquid sensor but brewing is finishing its last two minutes.

At any moment taking out the jug or pressing the button (either physically or remotely) the machine stops brewing coffee.

5.2.2 Authentication & Calendar API

The core component of our project is actually making the coffee machine *smart* by connecting it to the internet, and more specifically, to Google Calendar. We chose Google Calendar because of its widely used REST API, suitable for usage with Particle Webhooks, and because Google is a very popular provider of email and other services, e.g. a calendar, in general, so the chance that the potential users already have a Google account are quite high.

To use the Google Calendar API the user has to authenticate the device for it to get access to users personal information. On embedded devices with little input capabilities the OAuth 2.0 system lets devices generate a special code that the user will then provide on the website using other more input-capable device. The whole multi-step process is quite complex and involves three different Webhooks and a `Particle.Function` to successfully connect the device to the users Google Account.

1. The user remotely triggers a `Particle.Function` on the device.
2. The Photon publishes an event to the Particle Cloud to ask Google servers for the `device_code` needed for authentication.
3. On webhook response message, which the Photon is subscribed to, it gets the `device_code` and `user_code`. Then it starts counting for one minute while flashing the onboard led, and provides the user with the `user_code`.
4. The user then has one minute to navigate to `https://google.com/device` (preferably this should be done beforehand) and enters the `user_code` provided by Photon. Finally the user grants access to the calendar data to the Photon.
5. After the countdown the Photon publishes another event with the `device_code` as data, triggering a webhook which connects to Google's servers and asks, on the Photon's behalf, for `access-` and `refresh-tokens`.
6. Hopefully the Photon gets another response message from the webhook event, which it is subscribed to, containing `access-` and `refresh-tokens`. Then Photon saves them into its emulated EEPROM memory, to ensure that the refreshtoken is saved even if power is cut off.

The Access token is used for making calls to the API. But it is only valid for about one hour. After it expires the refresh token is used to request another access token. That's why once authenticated, the device only needs to store the refresh token and it doesn't have to bother the user to repeat the authentication process.

After some time when the access token has expired but before other requests to the API, the Photon just needs to publish an event to the Particle Cloud, and provide the refresh token as the data, which will trigger a webhook that will get a new access token and send it back to the Photon in a response message to the topic, which Photon is subscribed to.

5.2.3 Scheduling

An important part of our project is time scheduling. We want to make sure that when the event is fetched from the Calendar API the Photon will start brewing correctly about one hour before. Also we want to make sure that the event will be fetched at night, not during the day. To achieve this we use Software Timers. These are FreeRTOS timers, allowing up to 10 to run simultaneously. They take as arguments an `unsigned int` as a number of milliseconds and the function which will be triggered when the timer stops. When the device is authenticated every time it starts it checks if its night (between 1 and 6am UTC+1 time)[4] and if it is, to refresh its access token and then to retrieve the next event from the API, and to set a timer to trigger exactly one hour before. If it's not night, another timer is set to trigger after next midnight to fetch the next event by setting the first timer. When the event is triggered, that is the coffee starts brewing as scheduled, the timer for fetching new data at midnight is reset. And that's how this loop works indefinitely[5].

Also crucial to our operations is keeping track of the time in regards to time zones. Google Calendar uses RFC3339[6] date format. When we fetch the event from Google Calendar API we use Mustache[7] to get only the `startTime` of an event. Then we decode it by parsing the string and manually calculate the offset of how many hours, minutes and seconds we need to wait until brewing. We also need to convert this waiting time to milliseconds. Although the time of checking the first event is fixed to UTC+1 time, our algorithm for calculating waiting time works with any time zone.

6 Test/Verification

When the design of this project was finalized, it was divided into a series of interacting systems. Each of these systems was tested individually, with any issues within being solved or worked around before integration with the other systems involved. As with section 5 the details of these tests will be divided into the hardware and software sides of the project.

6.1 Hardware

The main component that required testing greater than just ensuring basic functionality was the FSR. As previously stated in order to use it it must be in series with another resistor to form a voltage divider. The value of the most ideal resistor for this voltage divider was initially unknown. The ideal resistor

[4]We suppose the device is to be used in Denmark, so these hours are basically hard coded.

[5]The assumption we make is that the user won't have any events at night, between 1 and 6am.

[6]`http://tools.ietf.org/html/3339`

[7]`http://mustache.github.io`

would give the greatest difference in voltage between no load and the maximum load the sensor was needed to withstand. In order to find this, a potentiometer was used during testing of the FSR, setting it at various values and using a multimeter to measure the voltage across the potentiometer with given loads on the FSR. However, it soon became very apparent that the FSR used was designed for comparatively smaller loads. That is regardless of the resistance of the potentiometer, any load on the FSR would be read by the Photon as the maximum load. It is because of this that the FSR must be treated as essentially a digital switch, causing its location on the machine itself to be changed to reflect this.

The other two inputs —the liquid level switch and override button— needed only basic functionality testing, that is to make sure there was no physical issues with either. Reading an on/off switch with an Arduino like device such as the Photon is trivial (as stated a pull down resistor was used), so these needed no further testing.

Similar to the two basic inputs above, the status LED's needed no thorough testing, just simply making sure they worked was all. The only issue that came up in testing is that initially 10 kΩ resistors were used in series with the LEDs, this proved to be too high of a resistance as the LEDs were barely visible when lit. The resistors were lowered to a more typical 1 kΩ and this issue was solved.

The component that proved to be the most problematic, as previously hinted was the relay. When the relay was initially being tested, it was attached to the Photon as indicated below in Figure 1.

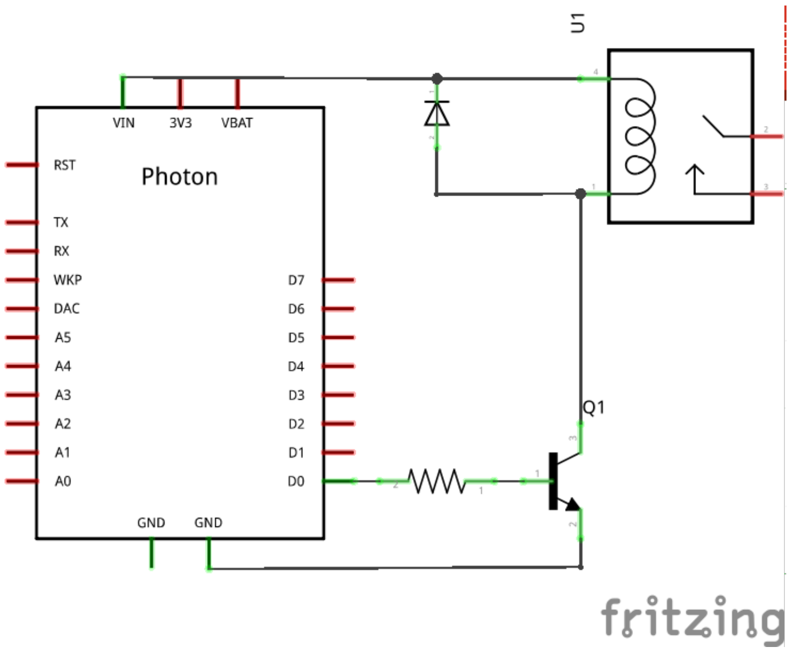

Figure 1: Initial relay control circuit

However, this circuit did not work. In testing, it was known that the Photon, resistor, transistor, and diode all functioned fine on their own. It was initially thought that the Photon may not be able to supply enough power to energize the coil of the relay. To test this, the coil was attached to a benchtop power supply capable of supplying 5V and up to 1A of power, well over the maximum current the relay needed based on the datasheet. This also proved to not work. The resistance of the coil itself was measured to ensure that there was a complete circuit between the two terminals, there was. Rather than continue to spend time trying to find some circuit to control the relay, alternatives were looked into. As previously stated, a solid state relay was chosen due to the significantly lower current and voltage requirements needed for it to function. This was tested by attaching it directly to the Photon, this was immediately successful, and this system was proven to work and was ready for integration with the rest of the project.

6.2 Software

Because our software consists of three distinct and independent problems —that is Safety features, Authentication & API, and Scheduling— we could decouple the testing and test each of these modules separately. For all the testing we used Serial monitor, to log what happened inside the machine.

Testing Safety features was easy and straightforward. First we checked if we can read the sensors and then we programmed the device so it would allow brewing only on certain conditions. After connecting the Photon to the coffee machine we just needed to test common scenarios like starting brewing coffee and taking the jug off, starting brewing without enough water etc. When we confirmed that our software worked we could integrate it with other parts of our project.

Authentication & Calendar API took us quite a bit of time to get right. This is due largely to the fact that we needed to get to know Google OAuth 2.0 authentication system. But when we setup everything testing was quite easy. First we were trying API functions by `curl` and then we just had to recreate them with Particle Webhooks. Testing fetching data was done by setting particular event in the calendar and comparing results of the data retrieved from API call.

The last piece of a puzzle was was Scheduling. When we implemented it, first we tested all the functions on very short intervals, like 2, 3 and 5 minutes (by setting special events in the calendar that we were fetching). Then we connected all our modules and made final test. We tried to use it as it is suppose to, we had just connected the device to our Google Calendar, set up specific event and see if on the next day it would brew the coffee. Apart from that the Photon was connected all night to the computer with Serial monitor turned on (the computer was set not to hibernate/sleep) so we could see logs of what exactly it was doing.

After confirming that all the functionalities are working as expected we started code cleaning and refactoring.

7 Conclusion

This project helped to show the design process of IoT devices, and the various considerations that must be made when creating them. This project however did not have as large of a focus on the traditional aspects of IoT device design, that is power usage and management as well as dealing with lapses in internet connection. It was mainly focused on integrating the Particle platform with existing services such as IFTT and a web API like Google Calender. The hardware side proved to be simple enough, and interfacing the software and hardware as easy as listing a number of pins and what is expected of each of them. Considerations did have to be made with regards to safety, but these proved easy enough and in a more refined version of the design these safety issues would be the same as any commercially available coffee machine.

8 Appendices

8.1 Appendix A, Hardware Schematic

Schematic of all hardware used in this project.

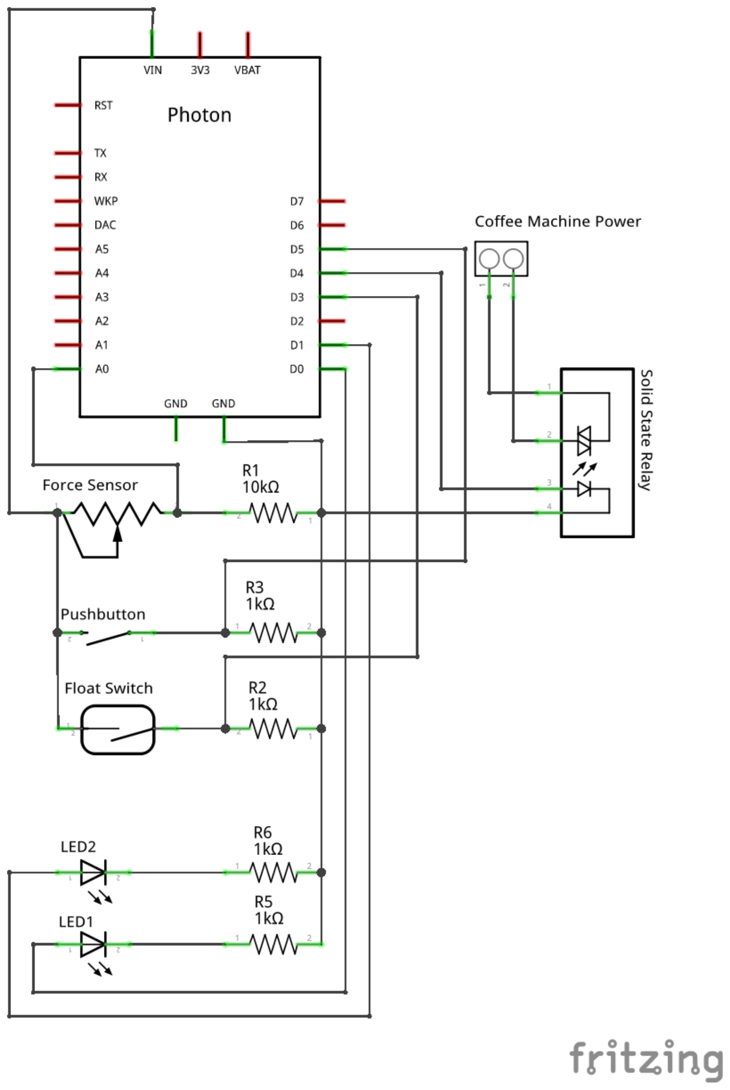

Figure 2: Hardware schematic

8.2 Appendix B, Hardware List

All hardware used in the project based off of the schematic is listed below. The part labeled *Coffee Machine Power* on the schematic in section 8.1 represents the terminal blocks built into the solid state relay, and as such is not included in the table. Part numbers are provided when appropriate.

Part Name	**Part Number**	**Description**	**Quantity**
R1	N/A	10 kΩ Resistor	1
R2-R6	N/A	1 kΩ Resistor	5
Pushbutton	N/A	SPST Momentary Button	1
LED 1-2	N/A	Green 5mm LEDs	2
Float Switch	311120001	Liquid Float Switch	1
Force Sensor	TPE-503B	Resistive Force Sensor	1
Solid State Relay	ESR5102402500	High Power Solid State Relay	1