Automatic Generator Battery Backup Controller

Zac M. O'Dell, Jonathan A. Gibson, Joshua P. Adair, Naveen M. Chulani, Clinton C. Iyizoba, Fahmi I. Hashi, Jeff C. Tilley

***Abstract*—There's a need for automatic generator starter systems. Individuals who are dependent on solar power might rather not go outside to manually start their generator(s) everytime the battery connected to their house is providing insufficient power. Similar products/kits on the market can cost several hundred dollars. Because consumers would prefer to spend less money on a simple product, there is a benefit to making an automatic generator starter at a lower cost. The generator should be turned on when solar panels aren't producing enough energy to power a person's home. The automatic generator starter system would assist in providing sufficient power to an individual's home, complementing solar panels' inconsistent charging capabilities. The goal is to create an automatic generator controller at a lower cost than what is currently on the market, while also providing extra features.**

***Index Terms*—arduino, automatic, backup, battery, controller, electric, generator, panels, solar.**

I. INTRODUCTION

THIS section discusses prospective project devices for the hardware system implementation. It will cover generators, the chosen microcontroller and its peripherals, DC to DC converters, voltage regulators, relays, terminals, and diodes.

A. Literature Review

Many people have designed their own automatic generator starters/controllers. For the most part, all of the designs follow the same principles. A signal generally needs to inform a microcontroller if the generator is powered on or powered off. The microcontroller then sends an activation signal when the generator needs to be started. However, generators and microcontroller boards typically don't operate at the same voltage. Most of the input and output pins on a microcontroller are roughly 5 VDC, while generators output a different and much higher voltage (typically 120 VAC). Sending a signal much more than 5 VDC to the inputs of any microcontroller could badly damage it and other devices connected to it, depending on their specifications. Sending a 5 VDC signal to the generator starter will also not power it on. In order for a microcontroller to correctly communicate with the generator, a voltage regulator or a voltage divider needs to be used by converting the microcontroller output to the correct amount of voltage, for the generator starter.

In most projects, similar to this one, relays are also used as switches to update the status of the generator. The only major difference between this project and other ideas is that some other projects use different microcontrollers, such as Raspberry Pies. Besides this, all automatic generator starters/controllers are built using similar ideas and hardware.

B. Generator

A generator is a device that converts mechanical force into electrical current through a process known as electromagnetic induction. This mechanical force can be supplied by a number of external sources such as gasoline, wind, etc.

In order to start the generator, the starter and the on/off switch are needed. The starter button, once pressed for a few seconds, turns the pistons within the generator. The on/off switch ignites the gasoline within the generator as the pistons turn. This then provides the input mechanical power that will be converted to electric power. Within the generator, there are also two sets of windings/coils: the rotor windings and the stator windings. The rotor windings are wound around a rotating shaft. Once the shaft starts turning and an alternating current is supplied to the rotor windings, a rotating magnetic field is produced, which in turn induces an EMF in the stator windings. Thus, mechanical energy (from the rotating shaft) is converted to electrical energy in the stator windings. This electrical energy can then be used to increase the voltage of a battery. A relevant example of this application is the recharging of an external backup battery when weather conditions are no longer conducive for generating sufficient solar power, for an individual's home.

C. Microcontroller and Peripherals

Although there are many different kinds of microcontrollers and microcontroller attachments available to implement an automated hardware system, it can be challenging to find what will work best for any particular system. Factors like cost, ease of use, efficiency, computational power, electronic limitations, and programmability can influence the decision.

This work was sponsored and financially supported by Tennessee Technological University (TTU), the Electrical & Computer Engineering (ECE) Department, and Dr. Michael A. Baswell.

Z. M. O'Dell is with the Electrical Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: zmodell42@students.tntech.edu).

J. A. Gibson is with the Computer Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: jagibson44@students.tntech.edu).

J. P. Adair is with the Computer Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: jpadiar42@students.tntech.edu).

N. M. Chulani is with the Electrical Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: nmchulani42@students.tntech.edu).

C. C. Iyizoba is with the Electrical Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: cciyizoba42@students.tntech.edu).

F. I. Hashi is with the Electrical Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: fihashi42@students.tntech.edu).

J. C. Tilley is with the Electrical Engineering Department, Tennessee Technological University, Cookeville, TN 38501 USA (e-mail: jctilley42@students.tntech.edu).

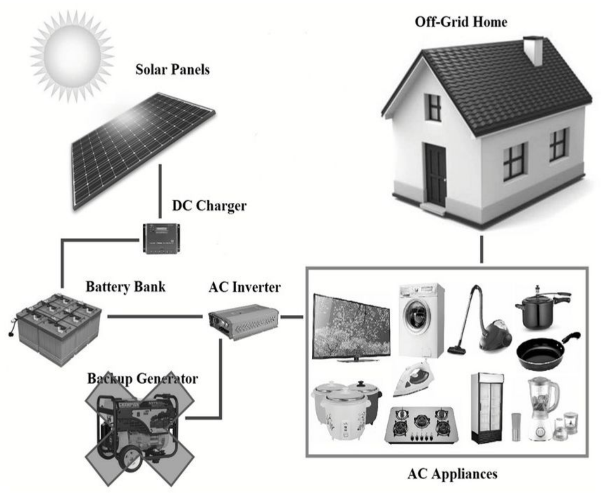


Fig. 1: Typical layout of "off-grid" power system operation [18].

1) Arduino Uno

Arduinos are microprocessing units that are programmed with *C* and *C++* programming languages and also provide input and output signals via analog, digital, and other pins/ports. Arduinos can serve as the brains of complicated devices by connecting and controlling many independent parts concurrently.

It is important to know how much power is required to power up an arduino properly. Because in many cases, Arduinos are connected to other devices, external power supplies and adapters connected to the Arduino need to supply at least 500 mA, or up to 1000 mA (1 ampère) for safe and proper operation of the Arduino and its connected devices. This is necessary to ensure enough power to make each component of the circuit function reliably. These important details are often printed directly on the adapter. Fig. 2 shows an image of an adapter ideally suited for powering an Arduino Uno. The most important information is underlined in red.

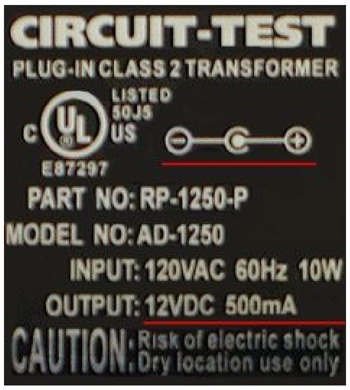

CIRCUIT-TEST
PLUG-IN CLASS 2 TRANSFORMER
C UL US LISTED 50JS
E87297
⊖—◐—⊕
PART NO: RP-1250-P
MODEL NO: AD-1250
INPUT: 120VAC 60Hz 10W
OUTPUT: 12VDC 500mA
CAUTION: Risk of electric shock Dry location use only

Fig. 2: Example of the information printed on the surface of an external power adapter appropriate for powering an Arduino Uno [5].

An Arduino's on-board regulator can, realistically, handle up to 20 VDC as an input voltage, so an Arduino hardware system can actually use an adapter that provides up to 20 VDC, but this is not particularly a good idea. The reasons why this is not good practice are twofold. First, most of the excess voltage supplied to the system will simply be lost as heat, which is terribly inefficient in any regard. Secondly, the on-board regulator will suffer if an excessive voltage is supplied to the Arduino. When excessive voltage is supplied to the Arduino, the on-board regulator actively steps down any high input voltage for use as 3.3 VDC and 5 VDC output pins. Over time, improper voltage supplies will overheat and damage the regulator, making the microcontroller inoperable. Therefore, it is a good idea to stick with 9 VDC or 12 VDC adapters for best results.

Arduinos have many useful on-board functions that make them flexible and dynamic. One such on-board device is an analog to digital converter (ADC). An ADC is a device that converts analog signals into digital signals. Arduino boards contain many of these ADC functions for use within the hardware program and digital output(s). The ADC modules are useful for converting continuous signals (sound, light, temperature, etc...) into discrete signals for use inside of a digital system.

2) Arduino Compatible Display

Typical Arduino compatible keypad shield displays include a 2 character × 16 character liquid crystal display (LCD) screen and 6 momentary push buttons to be mounted onto Arduino microcontrollers and/or screw shields for increased functionality. The keypad shield operates on 5 VDC and uses several pins to interface with the Arduino. A separate analog pin is used to read the signals from the momentary push buttons.

3) Screw Shield

A screw shield mounts on top of an Arduino, and allows the user to access all of the pins through screw terminals. Ultimately, this makes the connections more reliable. When mounting an LCD screen directly to an Arduino, the pins become inaccessible. When a screw shield is mounted between the Arduino and LCD screen, the pins become accessible.

D. Devices

This section will cover background information about DC to DC converters, voltage dividers, relays, terminals, and diodes. The operation of these devices are important to understand because they may be crucial to the design of the project.

1) DC to DC Converters

A DC to DC converter is a power electronics device that accepts a DC input voltage and then provides a scaled version of the input voltage as its DC output voltage. The output voltage can be greater or less than the input voltage (or equivalent if the device is a DC buffer). The converter output voltage is used to match the power supply requirements of the load. DC to DC converter circuits consist of a transistor or diode switch and energy storage devices like inductors and capacitors. DC to DC converters are typically used to provide a constant DC power supply to an electrical circuit. DC to DC converters whose output voltage is lower than the input voltage are commonly known as “buck converters”.

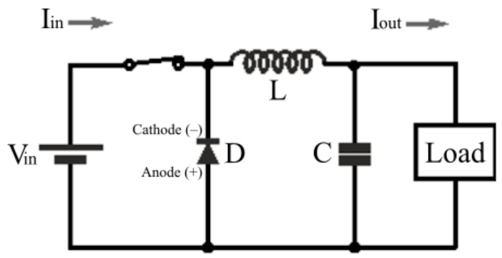


Fig. 3: Switch on period of a DC to DC converter [14].

When the switch is connected, as in Fig. 3, current flow to the load and capacitor is limited while energy builds up in the inductor. Charge build up on the capacitor and current flow to the load happens gradually. Also, the diode becomes reverse biased because of a large positive voltage on the cathode.

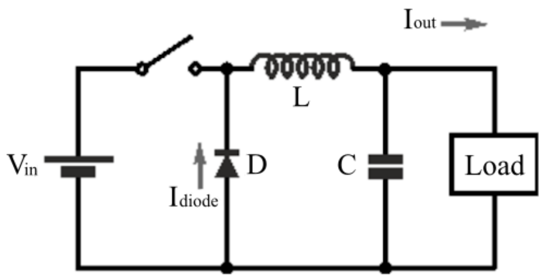


Fig. 4: Switch off period of a DC to DC converter[14].

When the switch is disconnected, as in Fig. 4, the energy stored in the inductor is released back into the circuit to keep the current flowing through the circuit until the next time the switch is connected. The diode, in this situation, becomes forward biased and current flow is directed towards the anode. When the stored energy in the inductor is released and starts to deplete, the load voltage begins to drop.

$$V_{out} = V_{in} \bullet \frac{t_{on}}{T} \quad (1)$$

The variable t_{on}, as shown in (1), is the “on-time” (how long the switch is connected) of the switching waveform and T is the periodic time of the switching waveform. By varying the mark to space ratio of the switching waveform, any output voltage within 0 VDC $\leq V_{OUT} \leq V_{IN}$ can be obtained.

2) *Voltage Dividers*

Another, and possibly easier, way of regulating voltage is by using a voltage divider. A voltage divider is a simple circuit which steps a large voltage down into a smaller one. Using just two series resistors, an input voltage, and a ground designation, an output voltage that is linearly proportional to the input can be realized. Voltage dividers are one of the most fundamental circuits in electronics. Calculating the output voltage of a typical voltage divider is shown in (2).

$$V_{out} = V_{in} \bullet \frac{R_2}{R_1 + R_2} \quad (2)$$

The implementation of such a voltage divider is shown in Fig. 5. It is important to note that if R_1 and R_2, in (2), have exactly the same value, in ohms (Ω), the voltage divider will effectively half the input voltage delivered to the output.

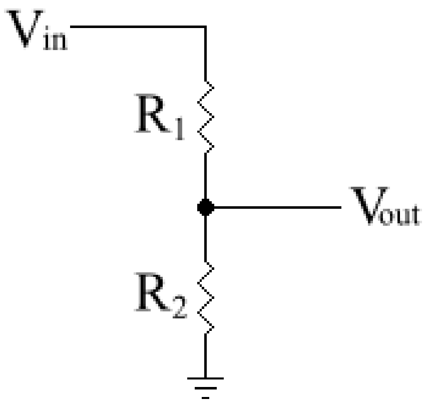


Fig. 5: Example of a common voltage divider circuit using two series resistors and a ground designation [16].

3) *Relays*

Relays are like switches that change states by applying a voltage to the signal input wire; the output connection is explicitly separate from the input connection. They are useful in applications where a system needs to detect a signal and allow the opening or closure of a specific junction. They are typically used in association with a sensor that will output a particularly useful value that a microcontroller is listening for. The microcontroller can then output a value based on the output of the sensor. An example of this application is an Arduino and a relay used to turn on a 120 VDC light bulb, shown in Fig. 6 below.

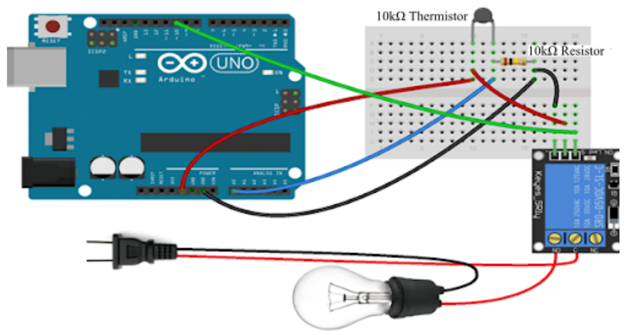


Fig. 6: A relay and an Arduino connected together to control/toggle a 120 VAC filament light bulb [9].

Relays have at least one normally open (NO), one normally closed (NC), and one common/ground (C) pin. A normally open pin is closed (shorted to the common node) when a 5 VDC input signal is applied. A normally closed pin is opened (disconnected from the common node) when a 5 VDC input signal is applied. An example of a 5 VDC relay, and is pin designations, is shown in Fig. 7.

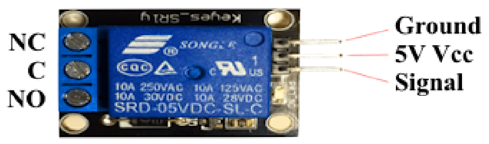


Fig. 7: 5 VDC Relay terminals and pins [9].

There also exist 120 VAC relays that will open or close depending on the existence of a 120 VAC signal at the input. Fig. 8 shows an example of such a relay. Fig. 8 depicts simulated terminal/pin numbers for demonstration purposes.

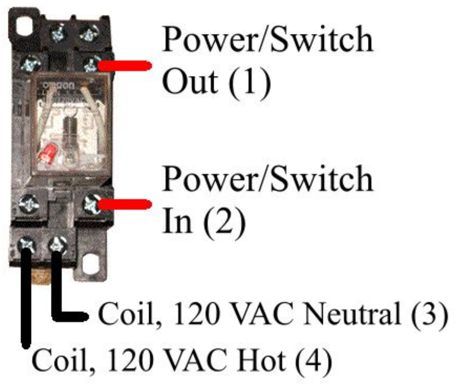


Fig. 8: 120 VAC signals are applied to the coil contacts shown with black wiring on terminals 3 and 4. The switched contacts are shown with red wiring on terminals 1 and 2. Terminals 1 and 2 are used as power in and power out to control a connected device [13].

4) Terminals

Terminals are electrical connectors that make it easier to join individual wires, as shown in Fig. 9. A wire screws down to one side of the terminal, and another wire screws down to the opposite side. Each side of the terminal is connected internally. Multiple terminals can be connected with jumpers to make a number of desired connections to a single node.

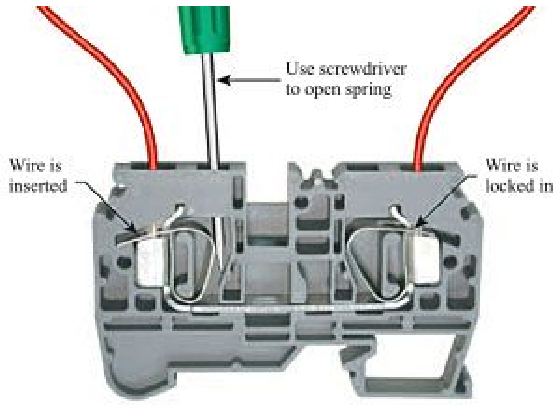


Fig. 9: Single terminal diagram [20].

5) Diodes

A diode is a semiconducting device that allows the flow of current in a particular direction. In particular, a light emitting diode (LED) is a type of diode that illuminates when an appropriate amount of current runs through it. LEDs have two leads, the anode and cathode, as shown in Fig. 10. The anode is generally the longer of the two leads. Current flows from the anode to the cathode. If too much current is applied, the LED can burn out. For this reason, a current-limiting resistor can be connected in series to the LED.

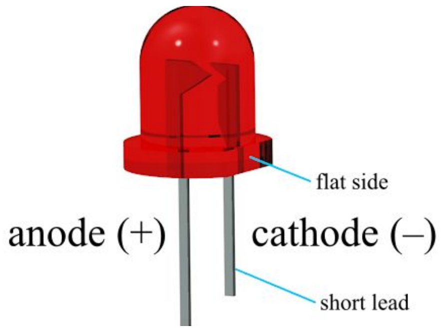


Fig. 10: Light emitting diode [19].

II. PROCEDURE

The Automatic Generator Battery Backup Controller (or simply BBC) is a hardware system regulated by an Arduino Uno microcontroller and controlled by an Arduino compatible LCD. The BBC system connects to general purpose generators and monitors an external battery's voltage. When the external battery's voltage drops below a user specified threshold (11.0V ~ 12.0V), the BBC system activates the generator's starter, for a user specified amount of time (2.0s ~ 3.0s), in an attempt to power on the generator. If the generator successfully starts and the external battery's voltage reaches a separate user specified threshold (13.0V ~ 14.0V), the BBC system powers off the generator and waits for the external battery's voltage to deplete again, then repeats. If the generator fails to power on, the BBC system delays for a user specified amount of time (20s ~ 60s), before activating the generator's starter again. The BBC system is only allowed three iterations of failed starter attempts in succession before displaying an error and taking appropriate action.

An Arduino compatible keypad shield display shows a main menu with live information about the external battery's voltage and allows the user to adjust the system parameters (minimum and maximum generator threshold voltages, the between-iteration delay time, and the generator's starter duration). If the attached generator does not power on within the allotted amount of starter time and iterations, the BBC system displays an error message ("Starter Failure!"), lights up the red LED indicator, and halts itself to be reset. The BBC system reset feature is to help prevent burn-out of the starter or generator during operation.

At any time while viewing the main menu, a user can toggle the system's automation status or adjust the system parameters. Using the “UP” and “DOWN” buttons on the LCD keypad allows the user to set/toggle the automation status, determining if the system should automatically try to start and stop the generator, or not. This feature is useful during system maintenance and any executive reasons for system stagnation. Using the “LEFT” and “RIGHT” buttons on the LCD allows the user to specify all the aforementioned system parameters.

These values are moderately variable to give the user a more dynamic and optimized controller for their particular generator. At any time within the parameter selection menu, the user can press “SELECT” to immediately leave the parameter selection menu and set all the parameter values to their current or changed states without needing to scroll through each parameter unnecessarily. The parameter selection feature also uses on-board Electronically Erasable Programmable Read-Only Memory (EEPROM) to read and set the parameter values. This feature persists parameter values through menu changes, system resets, and power outages to easily keep optimized settings.

The BBC system also has a feature that turns off the LCD display after 10 or more seconds of user inactivity (not pressing any of the buttons on the LCD keypad). When the LCD is off and the user presses “SELECT”, the LCD turns on and the system is returned to its current state. If the user has set the automation status to "ON", the system will run as normal in the background, even if the LCD is off. This inactivity feature is useful to save power/energy over the lifespan of the BBC system, to help prevent burn-out of the LCD, and to help prevent unintentional system manipulation from outside sources.

A. Hardware

For this project, a modular hardware design was implemented. The circuit is not enclosed, so the components can be easily accessed. Thus, the automatic generator starter can be easily repaired by replacing individual parts.

1) Construction

All of the different devices in the circuit were mounted on a metal plate. This was accomplished by drilling holes in the metal plate, then using nuts, bolts, and standoffs to hold the devices down. The devices held down directly to the metal plate are the Arduino, DC to DC regulator, two 5 VDC relays, and a Deutsches Institut fur Normung (DIN) rail. A screw shield and an LCD screen were mounted on top of the Arduino using its pins. Terminals and a 120 VAC relay were snapped onto the DIN rail. All electrical connections were made through the terminals, including wires, resistors, and an LED. For testing, a Tektronix PS283 power supply was used to simulate the battery, which was connected to the terminals. One end of a switch was connected to a 120 VAC power outlet to simulate the generator being off or on. The other end of the switch was connected to the 120 VAC relay. A image of the complete hardware system is included in the appendix.

The Arduino acts as the controller for the whole system. It is powered directly from the external battery. It reads the battery voltage through a voltage divider using an analog input. The Arduino controls the 5 VDC relays using two digital outputs, controls an LED using another digital output, and reads the generator status through the 120 VAC relay, using a digital input.

The DC to DC regulator can take any voltage from 7 VDC to 40 VDC and convert it to 5 VDC. In this case, it takes the battery voltage as its input and outputs a highly stable 5 VDC signal. The output will power the 5 VDC relays, and act as the reference voltage of the Arduino’s analog inputs.

The 5 VDC relays get their input voltage from the DC to DC regulator. They are controlled by the Arduino through their input pins. Because these are opto-isolation relays, the control circuits need little current, so the Arduino can control them directly. They are energized when the input receives a low signal. One relay, when energized, will connect a 12 VDC supply to the coil, and the other relay will connect a 12 VDC supply to the starter.

The 120 VAC relay is used to inform the Arduino about the status of the generator. The relay is energized when there is 120 VAC across the coil. The energized relay connects a ground to a digital input pin that is in pull-up mode.

Two of the resistors act as a voltage divider. The values of the two resistors were 15 kΩ and 75 kΩ. The voltage divider takes the battery voltage as its input, and outputs a lower voltage suitable for the Arduino to read through an analog input. Solar panels can reach a maximum voltage of 24 VDC. At this voltage, the output of the voltage divider is approximately 4 VDC (after being linearly scaled by 0.167 V/V), which is still safe for the Arduino to read. A third resistor, which is 200 Ω, was used to limit the current going through to the LED. The LED is connected to a digital output, and turns on when the generator fails to start after the third attempt.

The terminals were used to make the connections easier. All electrical connections were made through terminals. Only one wire was used on each side of each terminal. This allows the user to connect the battery and generator to the other side of the terminal strip on the outside of the device without touching the internal wiring. All the wiring to the Arduino and relays were on one side of the terminal strip. Fig. 11 shows a rough overview of the layout, construction, and wiring for the BBC system.

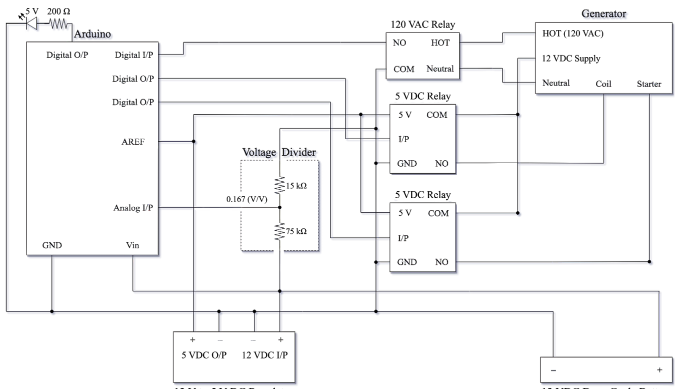


Fig. 11: Block diagram of the full BBC system.

2) Device Testing

All of the hardware was thoroughly tested to ensure proper functionality. The Arduino was used to test other devices by writing short programs for each. The power supply was set to 12 VDC and was connected to the input of the DC to DC regulator. A voltmeter was used to ensure that the output was 5 VDC. The output of the DC to DC regulator was used to power the 5 VDC relays. The inputs of the relays were connected to digital outputs. A short program was written to make the digital outputs alternate between high and low. Every time the signals were low, the relays were energized, connecting the common contact to the normally open contact. One end of a switch was connected to a 120 VAC outlet. The other end of the switch was connected to the input of the 120 VAC relay. When the switch was connected, the relay was energized, connecting the common contact to the normally open contact, and the Arduino "Serial.print();" command showed the input transition low. The individual resistors were connected to an ohmmeter to ensure they were the correct resistance values. A voltmeter was used to ensure the output of the voltage divider was between 0 VDC and 5 VDC, while the input ranged from 0 VDC ~ 20 VDC. The LED, with a 200 Ω resistor, was connected to a digital output. A short program was written to make a digital output high. This ensured that the LED would turn on when necessary. All hardware was ensured to operate properly before the software was implemented.

B. *Software*

The final result of the source code, and other resources, can be found at https://capstone-2-bbc.herokuapp.com and is also included in the appendix starting on page 8.

1) *Program Implementation*

The first part of the software implementation involved creating an generalized algorithm for the device. The first function that was implemented allowed the system to show the external battery voltage, read by the Arduino, onto the LCD shield display. Being able to see a live reading of the external battery voltage was an important first step to smoothly continue with the rest of the software implementation. The next function was to compare the voltage to certain levels that would trigger an action on the device and determine functionality. Timers were put up for the delays between each action of the device and was implemented next. There were several different timers that were added to the software after meeting certain functionality requirements. After these functions were successfully implemented, controlling the relays came next by setting certain relays for different conditions. The next functionality involved a menu system that allows the user to adjust the system parameters.

2) *Testing System Operation*

During the software testing phase of this project, an attempt was made to simulate every possible permutation of user and system input to fully test the software. The behavior of the system was observed during different voltage inputs using a Tektronix PS283 power supply, to simulate an external battery. Failed and successful generator starts were simulated using the custom generator simulator switch. Observations were made about how the system behaved after switching the generator simulator switch to "ON" or "OFF" at many predictable and unpredictable times, independently. Upon testing the BBC system, as best as possible, one software bug was not rectified.

C. *Software Issues*

Although the majority of the software works properly, if the generator dies shortly after starting and the external battery voltage is still below the minimum threshold, the BBC system only attempts to start the generator twice instead of three times, as specified in the project description.

III. COST ANALYSIS

For all hardware components used, the cost was estimated to build the BBC system in two different capacities: prototype construction and optimized consumer construction. Prototype total cost was calculated based on the assumption that no hardware components were previously owned and that every component must be purchased individually.

Optimized consumer total cost was calculated based on the assumption that all hardware could be bought at wholesale (bulk) price and that no testing components needed to be purchased or used. Wholesale quantity is upwards of 1000+ units for some components.

Actual prototype cost was free of charge. All hardware components were readily available and provided by Tennessee Technological University (TTU), the Electrical & Computer Engineering (ECE) department, and Dr. Michael A. Baswell.

TABLE 1
PROTOTYPE VS. OPTIMIZED CONSUMER HARDWARE COST

Category	Component	Quantity	Prototype	Consumer
Microcontroller	Arduino Uno	1 / 1	4.42000	4.42000
	Screw Shield	1 / 1	3.50000	3.50000
	LCD Display	1 / 1	4.11000	3.49000
Regulator	DC to DC Regulator	1 / 1	9.99000	9.99000
Relay	5 VDC Relay	2 / 2	2.14000	2.14000
	120 VAC Relay	1 / 1	15.70000	9.79000
Terminal	DK4N Terminal Blocks	100 / 28	41.21000	11.54000
Mounting	DIN Rail (*1.0m*)	1 / 1	7.86000	1.31150
	5 VDC Relay Board	2 / 2	2.89000	2.89000
	120 VAC Relay Socket	1 / 1	10.48000	6.11000
	Metal Base Plate	1 / 1	variable	variable
	Screws and Bolts	as needed	variable	variable
Power	Tektronix PS283	1 / 1	350.00000	N/A
	120 VAC 60Hz Socket	1 / 1	0.00000	N/A
Connectivity	Test Leads	2 / 2	12.18000	N/A
	Alligator Clips	2 / 2	1.04000	N/A
	75 kΩ ± 5% Resistor	1 / 1	0.10000	0.00784
	15 kΩ ± 5% Resistor	1 / 1	0.10000	0.00904
	200 Ω ± 1% Resistor	1 / 1	0.20000	0.02831
	Jumper Connections	100 / 26	27.98000	7.27480
	Copper Wiring	as needed	variable	variable
Miscellaneous	Red LED (*optional*)	1 / 1	0.46000	0.46000
	120 VAC 60Hz Switch	1 / 1	variable	N/A
Total ($USD)			494.36000	62.96149

Using the International System of Units (SI Units); V = volt, m = meters, Ω = ohm, kΩ = kiloohm, Hz = hertz.

Table 1 clearly shows the low cost implementation that was achieved, after the full construction of the BBC system.

IV. CONCLUSION

The objective of the project was to design an automatic battery backup controller that was less expensive than the current devices on the market. The final product is a fully functional Arduino-controlled automatic generator controller for an external backup battery, successfully priced below other products on the open market. This product can help those who depend on solar power, especially those in developing nations. Since the BBC system is modular, it can be easily repaired and modified. It eliminates the need to manually start and stop a generator, making the generator more convenient and efficient.

APPENDIX

All appendix content can be found following the references, starting on page 8. Additional appendix content includes AutoCAD pinout diagrams, software flowcharts, and the project source code.

REFERENCES

[1] Agarwal, Tarun. "DC to DC Converter Operating Principle and Functionality." *efxkits.co.uk*, 3 Jan. 2015, www.efxkits.co.uk/dc-dc-converter-operating-principle-functionality. Accessed 15 Feb. 2018.

[2] Burris, Matthew. "Learn More About Three Different Types of Voltage Regulators." *lifewire.com*, 4 Nov. 2017, www.lifewire.com/types-of-voltage-regulators-818851. Accessed 15 Feb. 2018.

[3] Coates, Eric. "Buck Converters." *learnabout-electronics.org*, 3 Sept. 2017, www.learnabout-electronics.org/PSU/psu31.php. Accessed 15 Feb. 2018.

[4] "Connecting your generator to your home", *powerequipment.honda.com* https://powerequipment.honda.com/generators/connecting-a-generator-to-your-home. Accessed 3 Apr. 2018.

[5] djmatic. "Arduino: What Adapter?" *playground.arduino.cc*, n.d., playground.arduino.cc/Learning/WhatAdapter. Accessed 3 Apr. 2018.

[6] user6263336. "Generator ATS" *hackster.io*, 5 May 2017 https://www.hackster.io/user6263336/generator-ats-72ae5f. Accessed 3 Apr. 2018.

[7] "Generator Data Sheets." *multiquip.com*, n.d., www.multiquip.com/multiquip/5051_ENU_HTML.htm. Accessed 3 Apr. 2018.

[8] "How does a generator create Electricity? How Generators Work." *dieselserviceandsupply.com*, n.d., http://www.dieselserviceandsupply.com/How_Generators_Work.aspx. Accessed 3 Apr. 2018.

[9] "How to Set Up a 5V Relay on the Arduino." *circuitbasics.com*, 11 Apr. 2017, www.circuitbasics.com/setting-up-a-5v-relay-on-the-arduino. Accessed 3 Apr. 2018.

[10] "Install a Transfer Switch and Beat the Next Blackout", *popularmechanics.com*, 2 Aug. 2013, https://www.popularmechanics.com/adventure/outdoors/tips/a9279/install-a-transfer-switch-and-beat-the-next-blackout-15762470. Accessed 9 Mar 2018.

[11] awaiskhawar. "Automatic Generater Starter." *instructables.com*, 27 Sept. 2017, www.instructables.com/id/Automatic-Generater-Starter. Accessed 9 Mar. 2018.

[12] Youngblood, Tim. "Make a Digital Voltmeter Using an Arduino." *allaboutcircuits.com*, 4 Jun. 2015, https://www.allaboutcircuits.com/projects/make-a-digital-voltmeter-using-the-arduino. Accessed 3 Apr. 2018

[13] "Our Do-It-Yourself Help Section." *hvacquick.com*, n.d., www.hvacquick.com/howtos/howto_relay.php. Accessed 9 Mar. 2018.

[14] Poole, Ian. "Step Down Buck Regulator / Converter." *radio-electronics.com*, n.d., www.radio-electronics.com/info/power-management/switching-mode-power-supply/step-down-buck-regulator-converter-basics.php. Accessed 15 Feb. 2018

[15] Chapman, Stephen. *Electric Machinery Fundamentals,* Mcgraw-Hill Education*,* 2005.

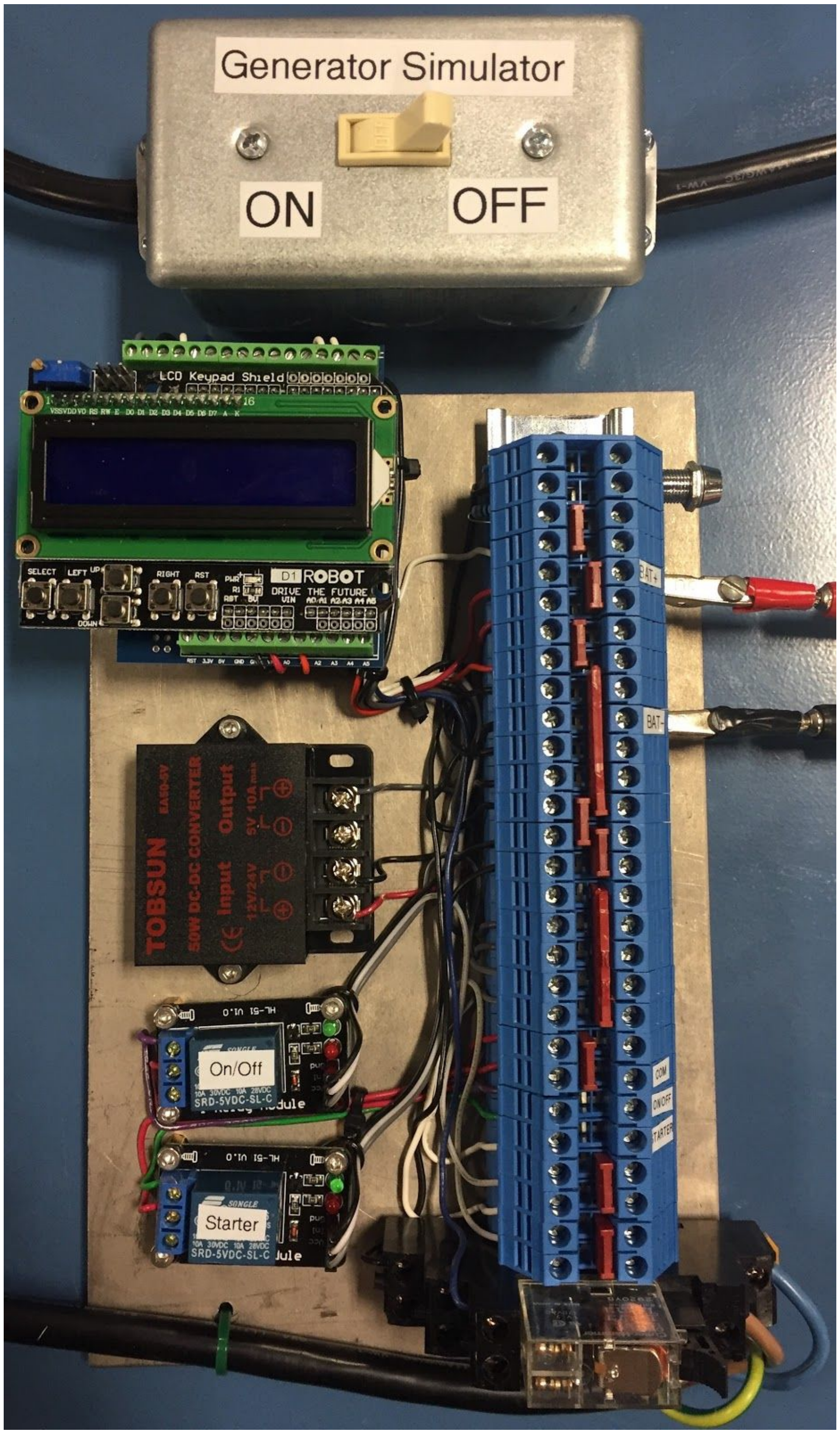
[16] Jimb0. "Voltage Dividers." *sparkfun.com*, n.d., learn.sparkfun.com/tutorials/voltage-dividers. Accessed 15 Feb. 2018.

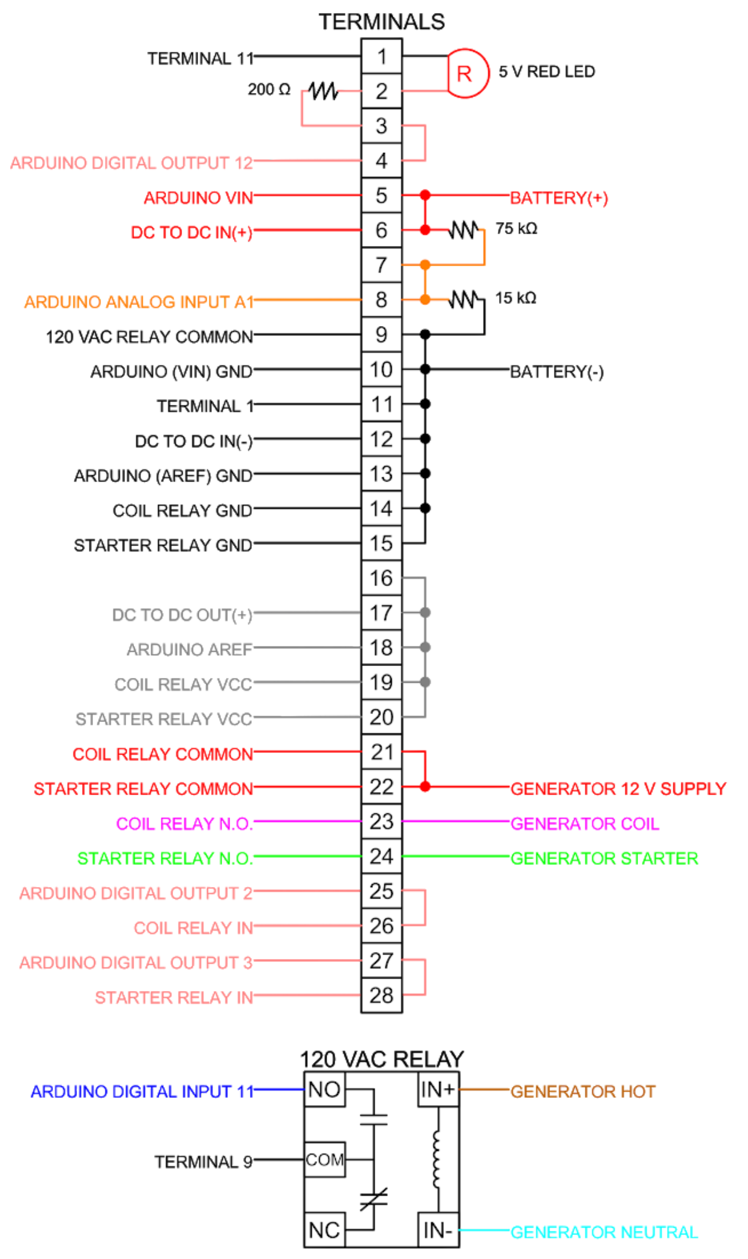
[17] "LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator", *fairchild semiconductor*, May 2006 http://hades.mech.northwestern.edu/images/6/6c/LM7805.pdf. Accessed 15 Feb. 2018.

[18] Miron Alexe, V. "Hydroelectric Backup System for Off-Grid Households." *Electronics, Computers and Artificial Intelligence,* July 2017, https://www.researchgate.net/profile/Viorel_Miron_Alexe. Accessed 8 Nov. 2018.

[19] Bharadwaj. "Light Emitting Diode Operation." Engineering Tutorial, 9 July 2016, https://engineeringtutorial.com/light-emitting-diode-operation. Accessed 08 Nov. 2018.

[20] "Terminal Blocks Information." Water Baths Information | Engineering360, https://www.globalspec.com/learnmore/electrical_electronic_components/connectors/terminal_blocks. Accessed 08 Nov. 2018.


Generator Simulator
ON
OFF
LCD Keypad Shield
SELECT
LEFT
UP
DOWN
RIGHT
RST
DFROBOT
DRIVE THE FUTURE
TOBSUN
50W DC-DC CONVERTER
Input 12V/24V
Output 5V 10A
On/Off
Starter
SRD-5VDC-SL-C
BAT+
BAT-
COM
ON/OFF
STARTER

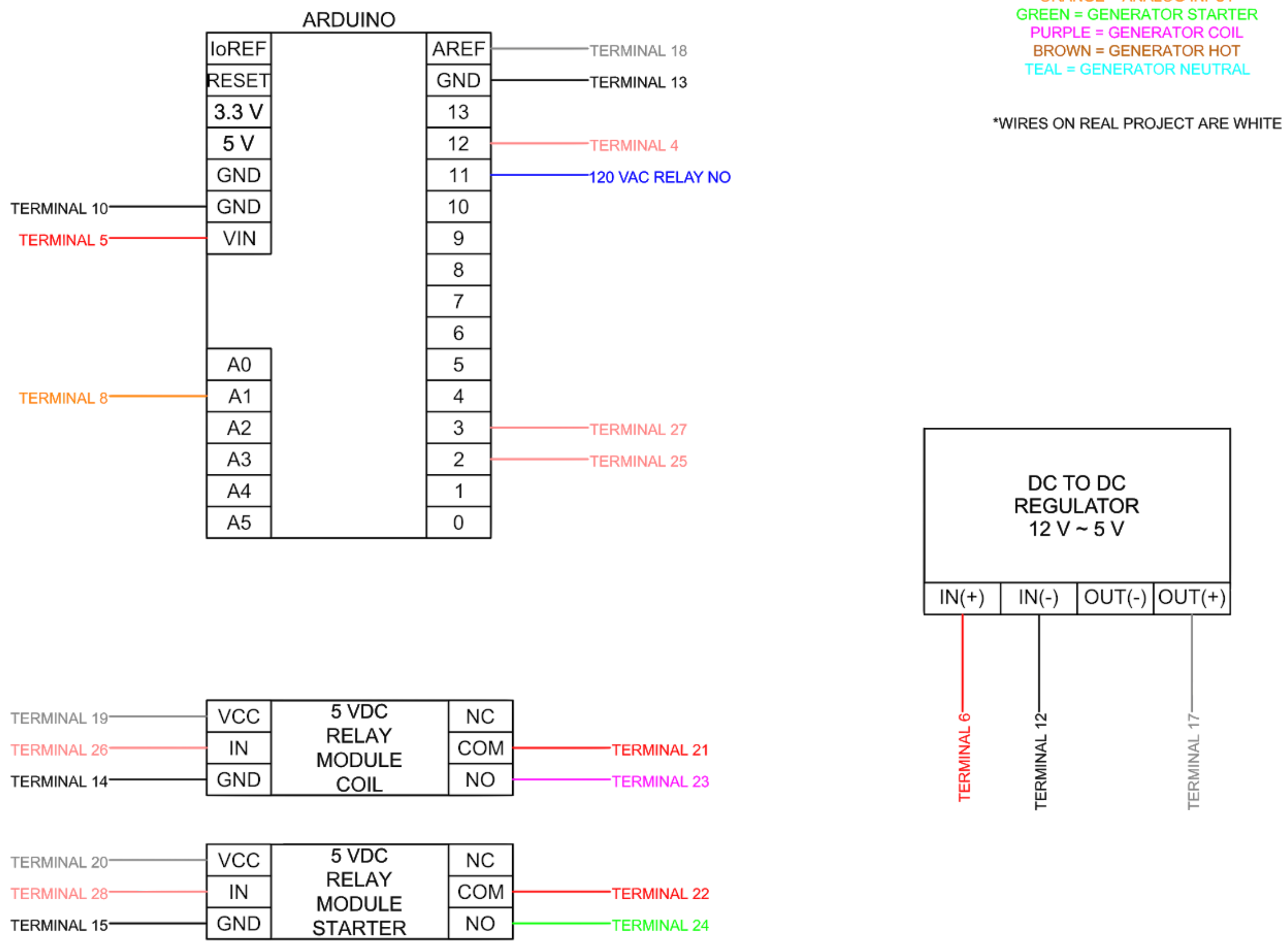


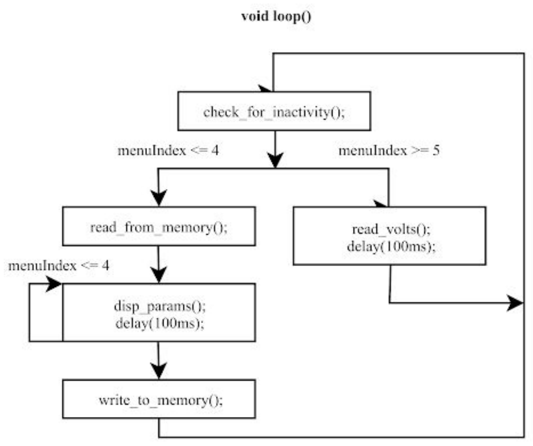
RED = 12 V
GRAY = 5 V
BLACK = GROUND
BLUE = DIGITAL INPUT
*PEACH = DIGITAL OUTPUT
ORANGE = ANALOG INPUT
GREEN = GENERATOR STARTER
PURPLE = GENERATOR COIL
BROWN = GENERATOR HOT
TEAL = GENERATOR NEUTRAL

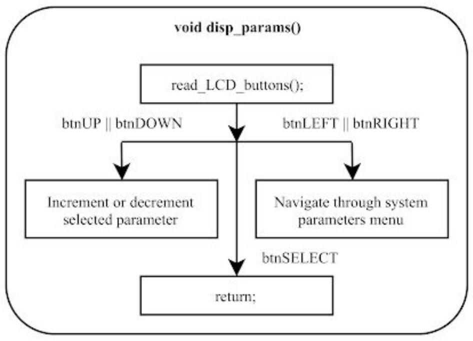
*WIRES ON REAL PROJECT ARE WHITE

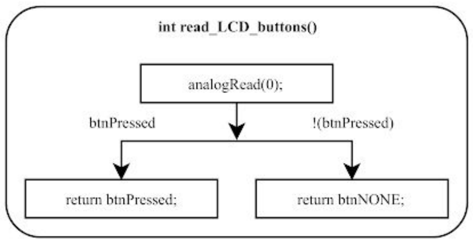
RED = 12 V
GRAY = 5 V
BLACK = GROUND
BLUE = DIGITAL INPUT
*PEACH = DIGITAL OUTPUT
ORANGE = ANALOG INPUT
GREEN = GENERATOR STARTER
PURPLE = GENERATOR COIL
BROWN = GENERATOR HOT
TEAL = GENERATOR NEUTRAL

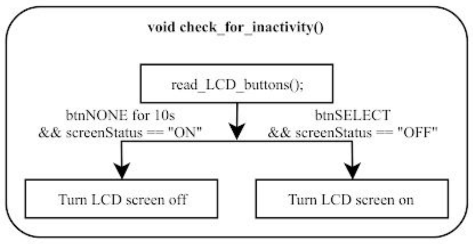
*WIRES ON REAL PROJECT ARE WHITE

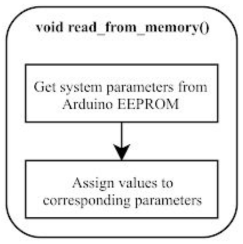


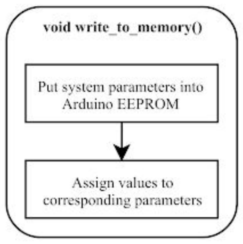

void loop()
check_for_inactivity();
menuIndex <= 4
menuIndex >= 5
read_from_memory();
read_volts(); delay(100ms);
menuIndex <= 4
disp_params(); delay(100ms);
write_to_memory();

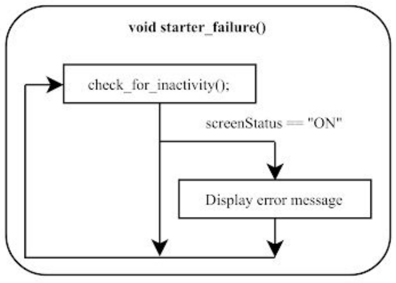

void disp_params()
read_LCD_buttons();
btnUP || btnDOWN
btnLEFT || btnRIGHT
Increment or decrement selected parameter
Navigate through system parameters menu
btnSELECT
return;

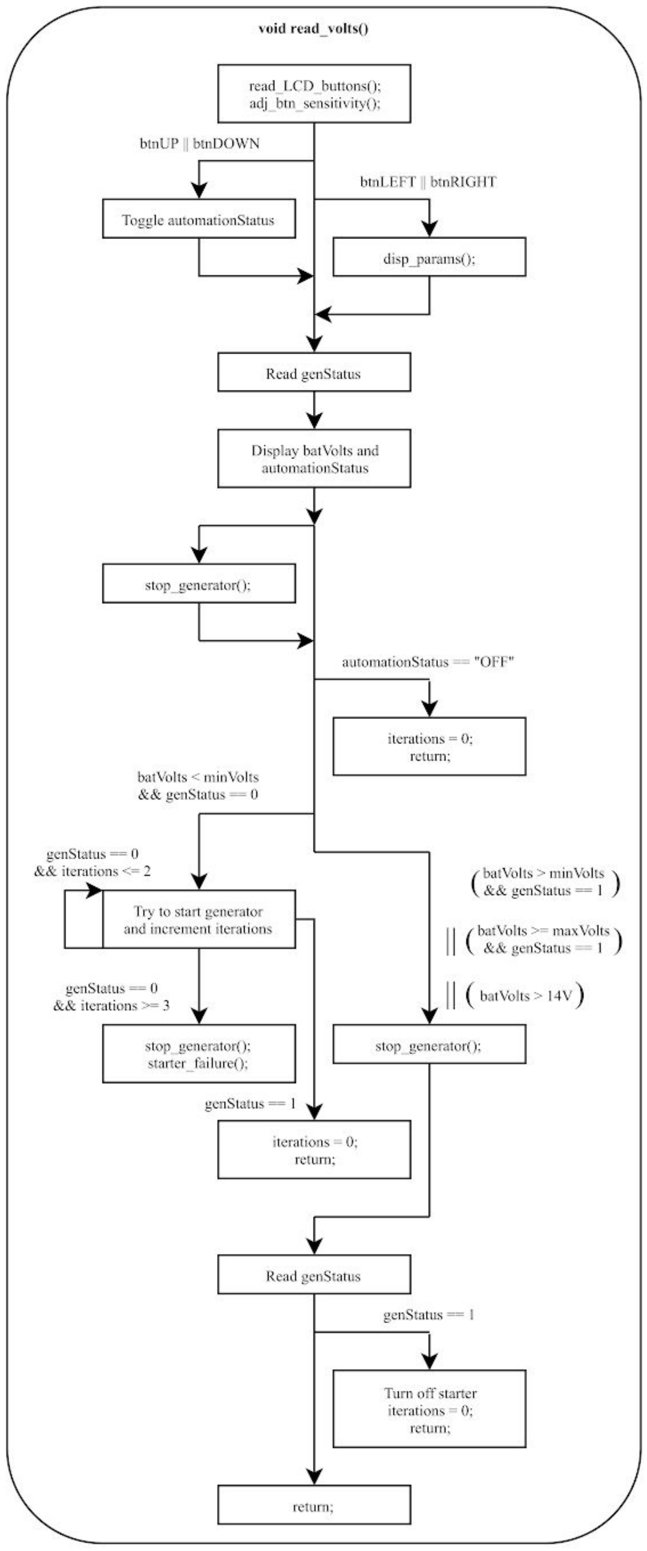

int read_LCD_buttons()
analogRead(0);
btnPressed
!(btnPressed)
return btnPressed;
return btnNONE;

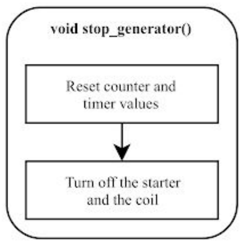

void check_for_inactivity()
read_LCD_buttons();
btnNONE for 10s && screenStatus == "ON"
btnSELECT && screenStatus == "OFF"
Turn LCD screen off
Turn LCD screen on

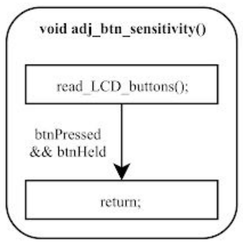

void read_from_memory()
Get system parameters from Arduino EEPROM
Assign values to corresponding parameters


void write_to_memory()
Put system parameters into Arduino EEPROM
Assign values to corresponding parameters


void starter_failure()
check_for_inactivity();
screenStatus == "ON"
Display error message


void read_volts()
read_LCD_buttons(); adj_btn_sensitivity();
btnUP || btnDOWN
btnLEFT || btnRIGHT
Toggle automationStatus
disp_params();
Read genStatus
Display batVolts and automationStatus
stop_generator();
automationStatus == "OFF"
iterations = 0; return;
batVolts < minVolts && genStatus == 0
genStatus == 0 && iterations <= 2
Try to start generator and increment iterations
(batVolts > minVolts && genStatus == 1)
|| (batVolts >= maxVolts && genStatus == 1)
|| (batVolts > 14V)
genStatus == 0 && iterations >= 3
stop_generator(); starter_failure();
stop_generator();
genStatus == 1
iterations = 0; return;
Read genStatus
genStatus == 1
Turn off starter iterations = 0; return;
return;


void stop_generator()
Reset counter and timer values
Turn off the starter and the coil


void adj_btn_sensitivity()
read_LCD_buttons();
btnPressed && btnHeld
return;

```
//include necessary libraries
#include <EEPROM.h>
#include <LiquidCrystal.h>

//setup LCD
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

//define LCD button values
#define btnRIGHT  0
#define btnUP     1
#define btnDOWN   2
#define btnLEFT   3
#define btnSELECT 4
#define btnNONE   5

//keep track of LCD button values
int lcd_key     = 0;
int adc_key_in  = 0;
int btnPressed = 1; //1 is btnPressed, 0 is btnNotPressed
int prevBtnPressed = 1; //1 is btnPressed, 0 is btnNotPressed
bool btnHeld = false; //default to false

//declare pin numbers for components
const int coil = 2;
const int starter = 3;
const int genStatus = 11; //if(genStatus == 0), then generator is on
const int redLight = 12;
const int backlight = 10;

//keep track of voltages and counters
int genStartIterations = 0; //number of starter iterations
int inactivityCounter = 0;
int rawVolts = 0;
int batMilliVolts = 0; //external battery voltage in mV
int mapValue = 31000;

//keep track of timers and delays
int timer2_3sec = 0;
int timerDelay = 0; //counter for the delay to reach "delayTime" value
int secondTimer = 0;
int starterTimer = 0;
const int mainDelay = 100; //universal delay for main loop

//keep track of the menuIndex
int menuIndex = 5;
#define changeMinVolt 1
#define changeMaxVolt 2
#define changeDelay 3
#define changeStartDelay 4

//default user specified system parameters
int minimumVoltage = 11500; //external battery minimum threshold
int maximumVoltage = 13500; //external battery maximum threshold
int delayTime = 300; //time between starter iterations
int genStartTime = 25; //time to crank generator starter

//keep track of EEPROM addresses
int addr = 0;

//keep track of generator signals
bool coilStatus = true;
bool starterStatus = true;
bool actualGenStatus;
bool voltageStatus = true;

//keep track of screen and automation status
char* automationStatus = "OFF";
char* screenStatus = "ON";

//---------------------------------------------------------------------------------------

void setup() //system setup
{
  //assign pin modes
  pinMode(coil, OUTPUT);
  pinMode(starter, OUTPUT);
  pinMode(genStatus, INPUT_PULLUP);
  pinMode(redLight, OUTPUT);
  pinMode(backlight, OUTPUT);

  //initialize baud rate and LCD
  Serial.begin(9600);
  lcd.begin(16, 2);

  //default "coil" and "starter" to off
  digitalWrite(coil, HIGH);
  digitalWrite(starter, HIGH);

  read_from_memory(); //get system parameters from EEPROM
}

//---------------------------------------------------------------------------------------

void loop() //main loop
{
  check_for_inactivity();

  if(menuIndex <= 4)
  {
    read_from_memory(); //get system parameters from EEPROM
    while(menuIndex <= 4)
    {
      disp_params(); //adjust system parameters
      delay(mainDelay); //main loop runs every 100ms
    }
    write_to_memory(); //put system parameters into EEPROM
  }
  else
  {
    read_volts(); //read external battery voltage
    delay(mainDelay); //main loop runs every 100ms
  }
}

//---------------------------------------------------------------------------------------

void read_volts()
{
  lcd.setCursor(0, 0);
  lcd_key = read_LCD_buttons();
  rawVolts = analogRead(A1); //read voltage from the generator
  batMilliVolts = map(rawVolts, 0, 1023, 0, mapValue); //read external battery in mV

  //dont allow rapid button changes
  adj_btn_sensitivity();
  if(btnHeld == true)
  {
    btnHeld = false;
    return;
  }

  //set "automationStatus"
  if((lcd_key == btnUP || lcd_key == btnDOWN) && screenStatus == "ON")
  {
    if(automationStatus == "ON")
      automationStatus = "OFF";
    else if(automationStatus == "OFF")
      automationStatus = "ON";
  }

  //adjust system parameters
  if(lcd_key == btnLEFT || lcd_key == btnRIGHT)
  {
    menuIndex = 1;
    disp_params();
  }

  actualGenStatus = !(digitalRead(genStatus));
  starterTimer++;
  secondTimer++;
  if (secondTimer == 10)
  {
    //display external battery voltage in V
    lcd.print("Battery = ");
    lcd.print(batMilliVolts / 1000);
    lcd.print(".");
    lcd.print((batMilliVolts % 1000) / 100);
    lcd.print("V  ");
    secondTimer = 0;
  }

  //display "automationStatus"
  lcd.setCursor (0, 1);
  lcd.print("Automation: ");
  lcd.print(automationStatus);
  lcd.print(" ");

  if(batMilliVolts > minimumVoltage && actualGenStatus == 0)
  {
    stop_generator();
  }

  //check "automationStatus"
  if(automationStatus == "OFF")
  {
    genStartIterations = 0;
    return;
  }

  //if the generator needs to be turned on
  if(batMilliVolts < minimumVoltage && actualGenStatus == 0)
  {
    if(voltageStatus == true) //avoid jitter
    {
      voltageStatus = false;
      minimumVoltage += 200;
    }

    if(coilStatus == true)
    {
      digitalWrite(coil, LOW); //sets coil hot at 12 volts
      coilStatus = false;
    }

    //while still trying to turn on generator
    while(actualGenStatus == 0 && genStartIterations <= 2)
    {
      actualGenStatus = !(digitalRead(genStatus)); //assign actual generator status
      digitalWrite(starter, LOW); //turn on starter
      starterStatus = false;

      if(timer2_3sec < genStartTime) //delay 2-3 seconds
      {
        timer2_3sec++;
        return;
      }

      digitalWrite(starter, HIGH); //turn off starter
      starterStatus = true;

      if(timerDelay < delayTime)
      {
        timerDelay++;
        return;
      }

      actualGenStatus = !(digitalRead(genStatus)); //read generator status
      if(actualGenStatus == 1)
      {
        genStartIterations = 0;
        return;
      }

      genStartIterations++; //go to next starter try

      //reset timer values
      timer2_3sec = 0;
      timerDelay = 0;
    }

    if(actualGenStatus == 0 && genStartIterations >= 3)
    {
      //failed to start generator
      stop_generator();
      starter_failure(); //stops the main loop
    }

    //reset starter iterations and timer
    genStartIterations = 0;
    starterTimer = 0;
  }
  else if(batMilliVolts >= maximumVoltage && actualGenStatus == 1)
  {
    stop_generator();
  }
  else if(batMilliVolts > minimumVoltage && actualGenStatus == 0)
  {
    stop_generator();
    if(voltageStatus == false) //avoid jitter
    {
      voltageStatus = true;
      minimumVoltage -= 200;
    }
  }
  else if(batMilliVolts > 14000) //failsafe
  {
    stop_generator();
  }

  actualGenStatus = !(digitalRead(genStatus)); //assign actual generator status
  if(actualGenStatus == 1) //if generator started
  {
    //prevent starter from getting stuck on
    digitalWrite(starter, HIGH); //turn off starter
    starterStatus = true;
    genStartIterations = 0; //reset starter iterations
  }
}

//-------------------------------------------------------------------------------------

void stop_generator()
{
    //reset counters and timers
    genStartIterations = 0;
    timer2_3sec = 0;
    timerDelay = 0;
    starterTimer = 0;

    //turn of generator
    digitalWrite(coil, HIGH);
    coilStatus = true;
    digitalWrite(starter, HIGH);
    starterStatus = true;
}

//-------------------------------------------------------------------------------------

void starter_failure()
{
  while(1) //loop forever
  {
    if(screenStatus == "ON")
    {
      //display error message
      digitalWrite(redLight, HIGH);
      digitalWrite(backlight, HIGH);
      lcd.display();
      screenStatus = "ON";
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Starter Failure!");
      lcd.setCursor(0, 1);
      lcd.print("Reset System...");
    }

    check_for_inactivity(); //dont display error forever
    delay(mainDelay); //delay 100ms
  }
}

//-------------------------------------------------------------------------------------

void disp_params()
{
  //reset and read LCD buttons
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd_key = read_LCD_buttons();
  stop_generator();

  //dont allow rapid button changes
  adj_btn_sensitivity();
  if(btnHeld == true)
  {
    btnHeld = false;
    return;
  }

  //exit if user presses "btnSELECT"
  if(lcd_key == btnSELECT)
  {
    menuIndex = 5;
  }

  if(menuIndex == changeMinVolt) //first system parameter
  {
    //display "minimumVoltage" parameter
    lcd.print("Min Volt: ");
    lcd.setCursor(0, 1);
    lcd.print(minimumVoltage / 1000);
    lcd.print(".");
    lcd.print((minimumVoltage % 1000) / 100);
    lcd.print("V");

    if(lcd_key == btnUP) //increase "minimumVoltage" by 100mV
    {
      minimumVoltage = minimumVoltage + 100;
      if(minimumVoltage > 12000) //ceiling
      {minimumVoltage = 12000;}
    }
    else if (lcd_key == btnDOWN) //decrease "minimumVoltage" by 100mV
    {
      minimumVoltage = minimumVoltage - 100;
      if(minimumVoltage < 11000) //floor
      {minimumVoltage = 11000;}
    }
    else if (lcd_key == btnRIGHT) //go to next parameter ("maximumVoltage")
    {
      menuIndex++; //"menuIndex" = 2
    }
  }
  else if (menuIndex == changeMaxVolt) //second system parameter
  {
    //display "maximumVoltage" parameter
    lcd.print("Max Volt: ");
    lcd.setCursor(0, 1);
    lcd.print(maximumVoltage / 1000);
    lcd.print(".");
    lcd.print((maximumVoltage % 1000) / 100 );
    lcd.print("V");

    if(lcd_key == btnUP) //increases the "maximumVoltage" by 100mV
    {
      maximumVoltage = maximumVoltage + 100;
      if(maximumVoltage > 14000) //ceiling
      {maximumVoltage = 14000;}
    }
    else if (lcd_key == btnDOWN) //decrease "minimumVoltage" by 100mV
    {
      maximumVoltage = maximumVoltage - 100;
      if(maximumVoltage < 13000) //floor
      {maximumVoltage = 13000;}
    }
    else if (lcd_key == btnRIGHT) //go to next parameter ("delayTime")
    {
      menuIndex++; //"menuIndex" = 3
    }
    else if (lcd_key == btnLEFT) //go to previous parameter ("minimumVoltage")
    {
      menuIndex--; //"menuIndex" = 2
    }
  }
  else if(menuIndex == changeDelay)
  {
    //display "delayTime" parameter
    lcd.print("Delay Time: ");
    lcd.setCursor(0, 1);
    lcd.print(delayTime / 10); //display "delayTime" in seconds
    lcd.print("s");

    if(lcd_key == btnUP) //increase "delayTime" by 1 second
    {
      delayTime = delayTime + 10;
      if(delayTime > 600) //ceiling
      {delayTime = 600;}
    }
    else if(lcd_key == btnDOWN) //decrease "delayTime" by 1 second
    {
      delayTime = delayTime - 10;
      if(delayTime < 200) //floor
      {delayTime = 200;}
    }
    else if(lcd_key == btnRIGHT) //go to next parameter ("genStartTime")
    {
      menuIndex++; //"menuIndex" = 4
      lcd.clear();
    }
    else if(lcd_key == btnLEFT) //go to previous parameter ("maximumVoltage")
    {
      menuIndex--;  //"menuIndex" = 3
    }
  }
  else if(menuIndex == changeStartDelay)
  {
    //display "genStartTime"
    lcd.print("Gen Start Time: ");
    lcd.setCursor(0, 1);
    lcd.print(genStartTime / 10);
    lcd.print(".");
    lcd.print(genStartTime % 10);
    lcd.print("s");

    if(lcd_key == btnUP) //increase "genStartTime" by 0.1 seconds
    {
      genStartTime = genStartTime + 1;
      if(genStartTime > 30) //ceiling
      {genStartTime = 30;}
    }
    else if(lcd_key == btnDOWN) //decrease "genStartTime" by 0.1 seconds
    {
      genStartTime = genStartTime - 1;
      if(genStartTime < 20)//floor
      {genStartTime = 20;}
    }
    else if(lcd_key == btnRIGHT) //go to main menuIndex
    {
      menuIndex++; //"menuIndex" = 5
      lcd.clear();
    }
    else if(lcd_key == btnLEFT) //go to previous parameter ("delayTime")
    {
      menuIndex--; //"menuIndex" = 3
    }
  }
}

//-------------------------------------------------------------------------------------

void write_to_memory() //put system parameters into EEPROM
{
  int addr = 0;
  EEPROM.put(addr, (float)minimumVoltage / 1000);

  addr = 5;
  EEPROM.put(addr, (float)maximumVoltage / 1000);

  addr = 10;
  EEPROM.put(addr, (float)delayTime / 10);

  addr = 15;
  EEPROM.put(addr, (float)genStartTime / 10);
}

//-------------------------------------------------------------------------------------

void read_from_memory() //get system parameters from EEPROM
{
  float fminv = 11.9f;
  float fmaxv = 13.4f;
  float fdelt = 2.00f;
  float fgensd = 5.00f;
  int address = 0;

  address = 0; //address 0 has "minimumVoltage"
  EEPROM.get(address, fminv);
  minimumVoltage = (int)(fminv * 1000);

  address = 5;
  EEPROM.get(address, fmaxv);
  maximumVoltage = (int)(fmaxv * 1000);

  address = 10;
  EEPROM.get(address, fdelt);
  delayTime = (int)(fdelt * 10);

  address = 15;
  EEPROM.get(address, fgensd);
  genStartTime = (int)(fgensd * 10);
}

//-------------------------------------------------------------------------------------

```

```
int read_LCD_buttons() //read LCD buttons
{
 adc_key_in = analogRead(0); //read the value from the sensor
 if (adc_key_in > 1000) return btnNONE;
 if (adc_key_in < 50)   return btnRIGHT;
 if (adc_key_in < 250)  return btnUP;
 if (adc_key_in < 450)  return btnDOWN;
 if (adc_key_in < 650)  return btnLEFT;
 if (adc_key_in < 850)  return btnSELECT;
 return btnNONE; //default to return btnNONE
}

//--------------------------------------------------------------------------------------

void adj_btn_sensitivity() //dont allow rapid button changes
{
  if(lcd_key == btnNONE)
  {
    prevBtnPressed = btnPressed;
    btnPressed = 1;
  }
  else
  {
    prevBtnPressed = btnPressed;
    btnPressed = 0;
  }

  if(prevBtnPressed == 0 && btnPressed == 0)
  {
   btnHeld = true;
  }
}

//--------------------------------------------------------------------------------------

void check_for_inactivity() //turn off LCD screen after 10 seconds of inactivity
{
  lcd_key = read_LCD_buttons();
  if(lcd_key == btnNONE && inactivityCounter < 100 && screenStatus == "ON")
  {
    //one loop of inactivity
    inactivityCounter++;

    if(inactivityCounter >= 100) //inactivity threshold reached
    {
      digitalWrite(backlight, LOW); //turn screen off
      lcd.noDisplay();
      screenStatus = "OFF";
      inactivityCounter = 0;
    }
    else
    {
      digitalWrite(backlight, HIGH); //turn screen on
      lcd.display();
      screenStatus = "ON";
    }
  }
  else if(lcd_key == btnSELECT)
  {
     digitalWrite(backlight, HIGH); //turn screen on
     lcd.display();
     screenStatus = "ON";
  }

  if(lcd_key != btnNONE)
  {
    inactivityCounter = 0; //reset inactivity timer
  }
}

//--------------------------------------------------------------------------------------
```

Zachary M. O'Dell was born in Clarksville, TN, in 1996. He is currently a senior level student obtaining a B.S. degree in electrical engineering with minors in mathematics and business from Tennessee Technological University, Cookeville, TN, and plans to graduate in the Fall of 2018.

Mr. O'Dell was an intern at Nashville Electric Service in Nashville, TN, during the summer of 2018. He was a part of the testing department and has experience working at substations doing testing, maintenance, and learning about the power equipment. His academic interests include power, digital systems, and control systems. He plans to pursue these interests both personally and professionally after his undergraduate career.

Jonathan A. Gibson was born in Nashville, TN, in 1996. He is currently a senior level student obtaining a B.S. degree in computer engineering with a minor in mathematics from Tennessee Technological University, Cookeville, TN, and plans to graduate in the Spring of 2019. He will be pursuing an M.S. in computer science from Tennessee Technological University, Cookeville, TN, following his undergraduate career.

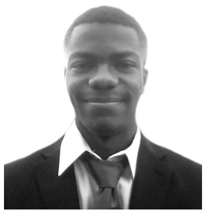
Mr. Gibson was an intern at Oak Ridge National Laboratory, Oak Ridge, TN, during the summer of 2017, studying artificial intelligence and natural language processing. He was also an intern at N2 Publishing, Wilmington, NC, during the summer of 2018, as part of a web design and maintenance team. His academic interests include artificial intelligence, quantum computing & theory, robotics, and web design. He plans to pursue these interests both personally and professionally after his graduate career.

Joshua P. Adair was born in Baliuag, Philippines, in 1995. He is currently a senior level student obtaining a B.S degree in computer engineering from Tennessee Technological University, Cookeville, TN, and plans to graduate in Fall 2018.

Mr. Adair is currently an intern at Tandus Centiva while finishing his B.S degree. He has created a web application that allows the company's project managers monitor data and automate reports. He is currently working on wireless power transfer research with Dr. Bhattacharya at Tennessee Technological University.

Naveen M. Chulani was born in Jamaica, in 1994. He is currently a senior level student obtaining a B.S. degree in Electrical Engineering with a concentration in mechatronics from Tennessee Technological University, Cookeville, TN, and plans to graduate in the Fall of 2018.

Mr. Chulani is currently assisting CEROC (Cybersecurity Education, Research and Outreach Center) on a research project that incorporates a Raspberry Pi controlled 6 DOF robotic arm. His academic interests include robotics, automation, and control systems. He plans to pursue these interests both personally and professionally after his graduate career.



Clinton C. Iyizoba was born in Enugu, Nigeria, in 1997. He is currently a senior level student obtaining a B.S. degree in electrical engineering at Tennessee Technological University, Cookeville, TN, and plans to graduate in the Fall of 2018. His academic interests include power systems engineering with a particular focus on renewable energy.

Fahmi I. Hashi was born in Somalia, in 1988. He is currently a senior level student obtaining a B.S. degree in electrical engineering with a minor in mathematics from Tennessee Technological University, Cookeville, TN, and plans to graduate in Fall 2018. Mr. Hashi was an intern at Mesa Associates during the summer of 2018. His academic interests include power system engineering.

Jeffery C. Tilley was born in Harriman, TN, in 1996. He is currently a senior level student obtaining a B.S. degree in electrical engineering with a minor in mathematics from Tennessee Technological University, Cookeville, TN, and plans to graduate in the Fall of 2018.

Mr. Tilley was an intern at Mesa Associates during the summers of both 2017 and 2018. His academic interests include power and control systems.