Senior Design Final Report

May 8th, 2022

Bird Bath Automation Controller (BBAC)

Team Banana Bath

Brandon Banarsi, Brandon Guzy, Vanessa Li

Faculty Advisor: Scott Tierno

Table of Contents

Executive Summary

The Bird Bath Automation Controller (BBAC) is an efficient way to routinely clean and fill a bird bath. For this project, an embedded system using the AVR128 microcontroller is being implemented to automate simultaneous cleaning and filling of two bird baths. The main function of the BBAC will be to perform a cleaning event, where the basin of the bird bath is flushed, and then refilled with clean water. A new printed circuit board (PCB) is to be designed and assembled with an upgraded AC-powered supply. In addition, AC solenoids will be used. There will be a new LCD interface that will be low-powered and have additional user activated pushbutton switches (PBSWs). A third valve is to be installed and controlled to regulate the flow of water to the top bird bath. A Wi-Fi component will be included in the system for easier access to the controls from a webpage on the local network. This allows the user to access the system's controls from anywhere the Wi-Fi reaches on a mobile device or computer, with no need to access the control box.

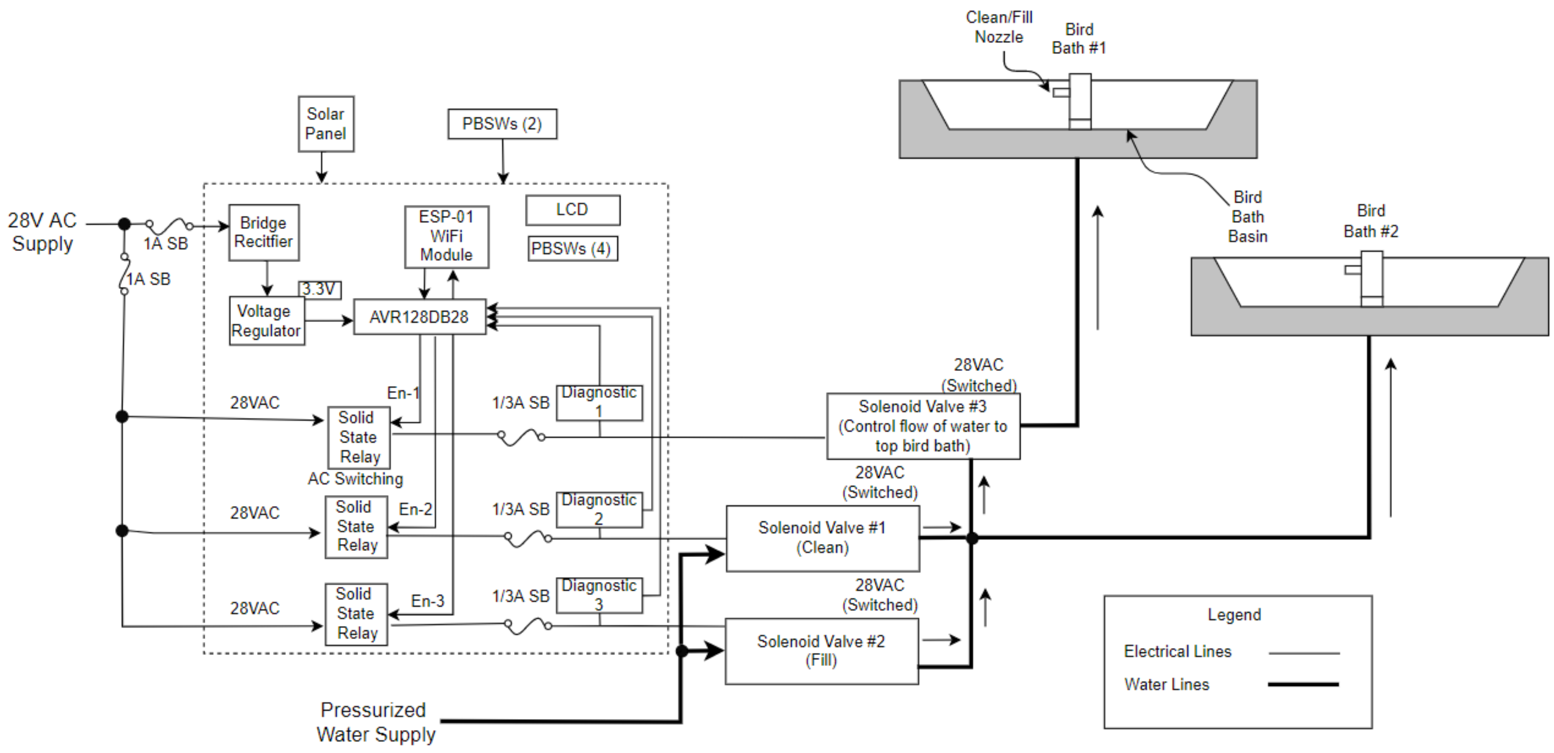


Figure 1: Block diagram of the entire bird bath system.

Background

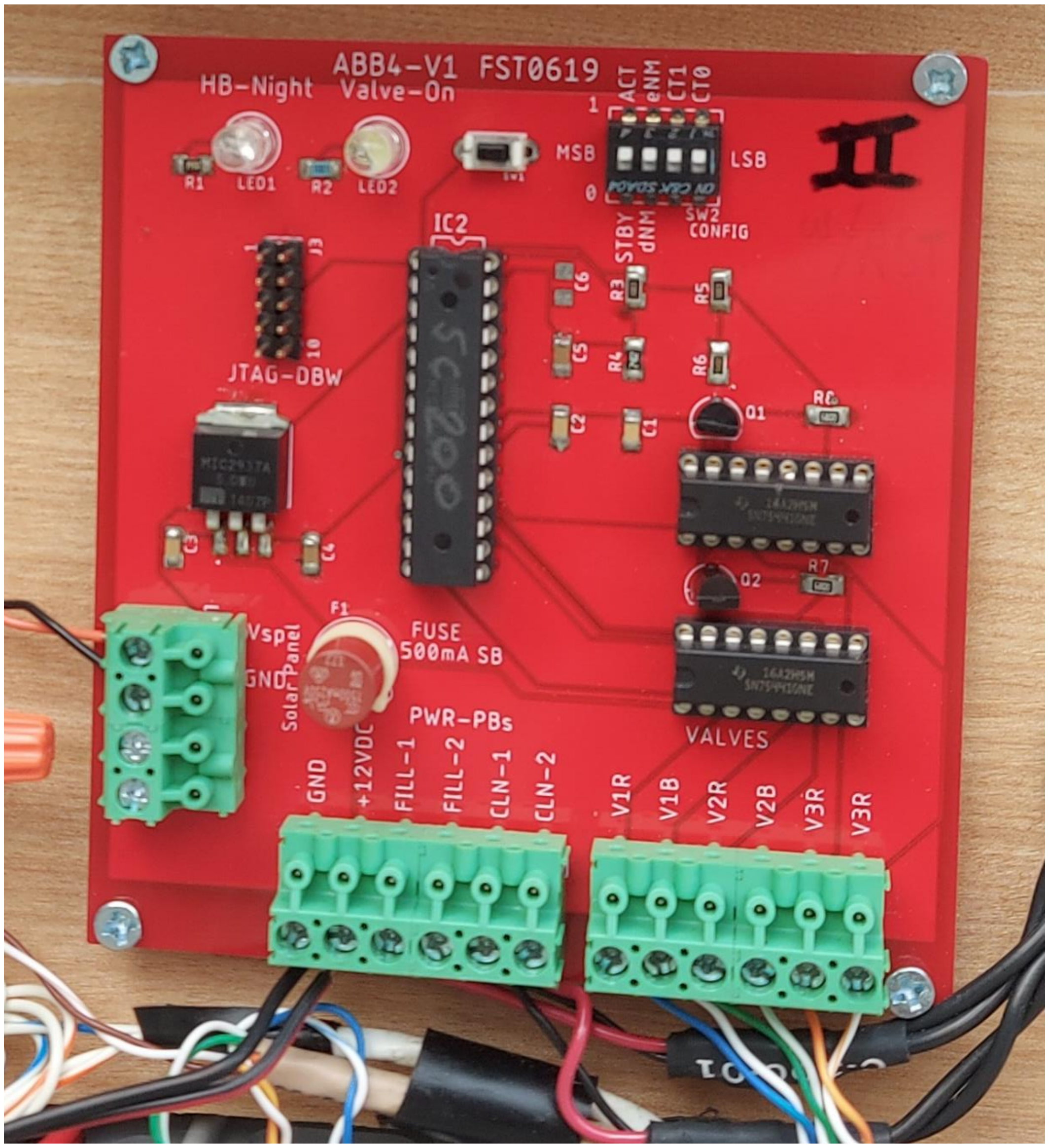


Figure 2: A photo of the previous BBAC PCB installation

Previous System

This project has upgraded an existing bird bath installation; the plumbing system and pipes were already installed. The existing bird bath used DC solenoid valves to control the flow of water. The updated uses AC solenoid valves to fix the sticking issues that were present in the DC system. To control the solenoid energizing current, solid-state relays (SSRs) are utilized.

For the interface, the existing system used push buttons and DIP switches for settings with no LCD for information display, information is instead conveyed with simple LED lights. An AVR microcontroller is used for the automation of the system, a button press can initiate a refill or cleaning cycle of the system. The microcontroller then starts a timer and runs the cycles again periodically. This existing PCB design can be seen in Figure 2, which shows the connections, microcontroller, LEDs, and DIP switch settings system.

Project Statement & Goals

The main goals of this project were to upgrade an existing bird bath automation system to use AC solenoids for increased reliability, to add a new interface with an LCD for managing the system, and a new Wi-Fi component for remote management. The upgrades for the system allow the user to see what is happening. As stated, the previous design did not have an LCD so it was difficult to tell when the next cycle would be or how long the clean/fill cycle has been going. With the LCD, this information is available to the user. In addition, the added Wi-Fi component allows for remote control of the system. Now, if the user was not able to get to the main control unit but still wanted to see how long until the next cycle or perform a clean/fill cycle, this can all be done on a phone/computer.

Social, Societal, and Environmental Impacts

The improvements made with this bird bath project have the potential to impact the local community and environment in several ways. The primary impact will be on the environment surrounding the installation because it provides a positive living environment for the birds. The bird baths provide a source of drinking and bathing water for the birds that is always clean because of the self-cleaning goal of the project. This draws birds to the area, which affects the environment.

Having birds in the area has mostly positive effects on the environment and people in the area. They offer natural pest control, eating bugs and other small pests in the area. They can help pollinate plants and disperse seeds, which has a positive effect on the area. People in the area can also enjoy watching the birds in the area, which has a positive effect on the community. One potential negative effect of having birds in the area is that they can be noisy at times, but the positives outweigh this negative. Some may enjoy the sound of birds singing in the morning. [10]

The impact of the project itself, the bird baths and the moving parts, on the surrounding community will not be very large because it is being done in a private backyard. The project will provide fresh water to the bird baths and keep them clean, which has a positive aesthetic impact on the area it is installed. Overall, we believe that this project will have a positive impact on the surrounding environment and community where it is installed.

Constraints

The constraints for this project include budget, size, and time. These constraints had to be kept in mind throughout the design of the system. The budget constraint on this project was $115 per student, or $345 total, which was met for both semesters. The advisor of this project, Scott Tierno agreed to pay for some expenses like the second PCB revision and parts for building the PCBs. He also loaned hardware like the display during the design period, so the budget constraint was not very difficult for this team.

There is also a time constraint on the project, which is to complete the project by the end of the Spring semester. This is the most difficult constraint because of the need to design, manufacture, test, and integrate the system by the end of semester with no extensions. To account for this constraint, the team made milestones such as having a functional breadboard by the end of the fall semester and PCB designed by spring break. These goals helped focus the team's effort, even if some of the goals were delivered a bit late. There were several setbacks during the project, but they were solved and the BBAC has been finished and installed the on time.

Another constraint is space, the project needs to fit in the space of an existing installation box. It also must interface with existing components like the side buttons on the case. Keeping these size constraints in mind during the design is important for success installing the PCB in the existing space.

Best/Worst Case, Hardware Design Discussion

DOGM163 and OLED Display

The first display that was used for the initial prototype of the system was the DOGM163. This was 3-line 16-character display. As the project moved along, it was determined that the display was not large enough. The 16 characters were not enough for the purposes that was needed, especially for the pushbutton options on the last line of the display.

A graphical OLED was also considered, this would be the absolute best-case design because it could allow more complex and fancy graphics. This would, however, require more time to integrate and increase the complexity of the AVR program. Instead of adding this graphical display, time and effort was diverted to working on the Wi-Fi component. This is because the display on the PCB can only be used when physically close to the PCB. The Wi-Fi component is a much more convenient interface, users can access it anywhere, so more time will be spent on this component.

MIC2937A, LM7805, MAX5035 and TL2575HV

The initial choice for a voltage regulator was the MIC2937A which can output a 5VDC constantly with a rectified input voltage of around 26V with an output current of 750mA. However, after using a stepdown transformer that provided 25.5VAC, the rectified input to the MIC2937A was roughly 40VDC which satisfies the maximum conditions but not the active conditions – rendering this regulator useless in the design. The momentary solution was to use an LM7805 which provided the correct output voltage with nearly double the output current. However, the maximum rating for the LM7805 is an input voltage of 35V. Although the chip worked, it is guaranteed that if the BBAC continued to supply 40V to the LM7805 over an extended period, it would surely cause intermittent shutdowns, or even break the component all-together [5] [6].

After calculating current draw for the overall system, a voltage regulator with output current of 1A provided enough clearance for the system, considering that the ESP8266 drew the most current at 200mA peak during transmission. In-depth research highlighted the MAX5035 5.0V voltage regulator as the most ideal part. The MAX5035 had an input range from 7.5 to 76V and would steadily output 5.0V with a current tolerance up to 1A [7]. This voltage regulator, however, was not of high availability due to a national chip shortage and was not available in high enough quantity for testing and possible usage. The alternative approach was to use an Evaluation Kit provided by Maxim that used the MAX5035. The EV Kit was a module that could solidify the entire power supply circuit solely. The only drawback of using the kit was that each kit would be $37.56 and had a factory lead time of up to 6 weeks. Buying the chip and recreating the EV Kit was the next approach, which was very component and cost heavy.

An exhaustive electrical characteristic analysis of the BBAC circuitry revealed that 3.3V would be efficient and optimal to power all components. The MAX5035 3.3V version was a viable option, but in looking for more DC-DC modules and switching power supplies, the TL2575HV 3.3V voltage regulator was a better option. It had a maximum input voltage rating of 60V, enough to account for any voltage spikes from the main power supply and could output 3.3V with up to 1A current tolerance. The leading positive of this IC voltage regulator was that it only needed 4 external components to be functional, as opposed to the MAX5035 which needed at least 7 (the rest were optional) [7][8]. It was then concluded that the TL2575HV 3.3V voltage regulator would be the finalized option for the power supply.

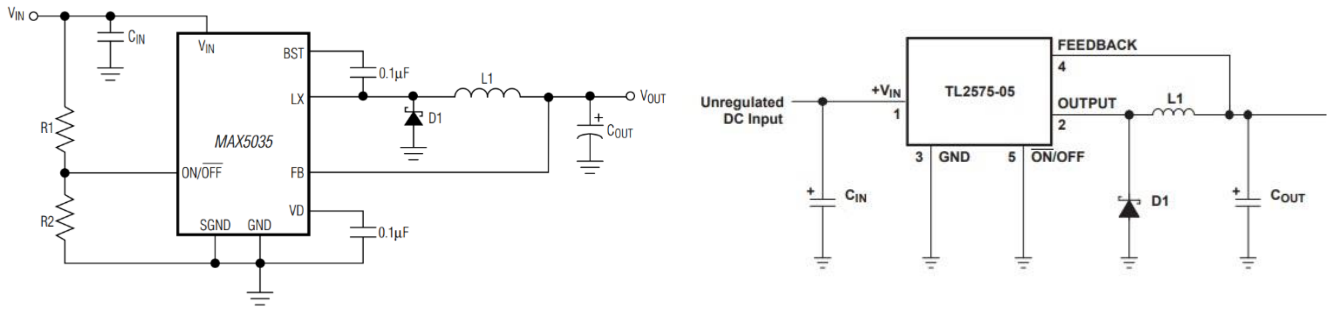


Figure 3: A side by side comparison of the MAX5035 (left) and TL2575HV (right) voltage regulators and their necessary external components [7][8]

MAX5035			**TL2575HV**		
Identifier	Value	Model	Identifier	Value	Model
R1	1M		Cin, Cout	100uF rated for 80V	107AXZ016MQ5
R2	384K		L1	330uH inductor rated for 1.5A	7447709331
D1	Schottky Diode	VS-50SQ100	D4	Schottky Diode	11DQ06TA
Cin	68uF	EEVFK2A680Q			
Cout	68uF	594D68X_010C2T			
Cbst	0.1uF	805			
L1	100uH	DO5022P-104			

Table 1: A side by side comparison of the necessary external components for the MAX5035 (left) and TL2575HV (right) voltage regulators corresponding to Figure X

SSR Diagnostic Circuit

The original design for the SSR diagnostic circuit revolved around a diode directing current to a voltage divider from the junction of the SSR output and a lead of the AC Solenoid. The diode connected directly to a capacitor that held charge steadily so that the voltage divider could provide a consistent voltage output to the analog input and ADC of the AVR128. Through primitive testing and experiments from the fall semester, it held true that a voltage on the SSR could be detected by the AVR128 through the diagnostic circuit. The removal of a fuse in-line with the SSR output and the AC Solenoid, before the diode, would simulate an open circuit and cause the solenoid to stop working. The AVR128 was able to detect such open and any subroutine for the diagnostic would show low voltage on that line and a failure. There was one diagnostic circuit per SSR.

However, an issue arose when testing the diagnostic circuit with the actual solenoids. The solenoids, being AC, acted as a wire that connected the diode and diagnostic circuit to the relative negative side of the AC Voltage supply. This allowed current to continuously pass through the diagnostic circuit and created a closed loop between the diagnostic circuit and the

AC Solenoid. Thus, the removal of the in-line fuse, and even the entire SSR, was not flagged on the AVR128, even though the functionality of the solenoid was dependent on the SSR and fuse.

To combat the issue, a second set of SSRs are used per solenoid to control the relatively negative side of the solenoid. A constraint of the system, being the lack of GPIO pins on the AVR128, prohibited a separate control signal for the SSRs on the relatively negative side of the AC power supply. Therefore, the diagnostic circuit can test the AC power supply through the solenoid and can detect low voltage but is rendered useless at determining a blown fuse. An LED was then placed after the diode in the original circuit, in parallel with the capacitor, and is only illuminated when the AC Solenoid successfully triggers. If there is a blown fuse or a solenoid issue, the user can determine which fuse and solenoid combination is faulty by noticing which LED does not turn on when it should.

Software Design Discussion

Software Design Overview

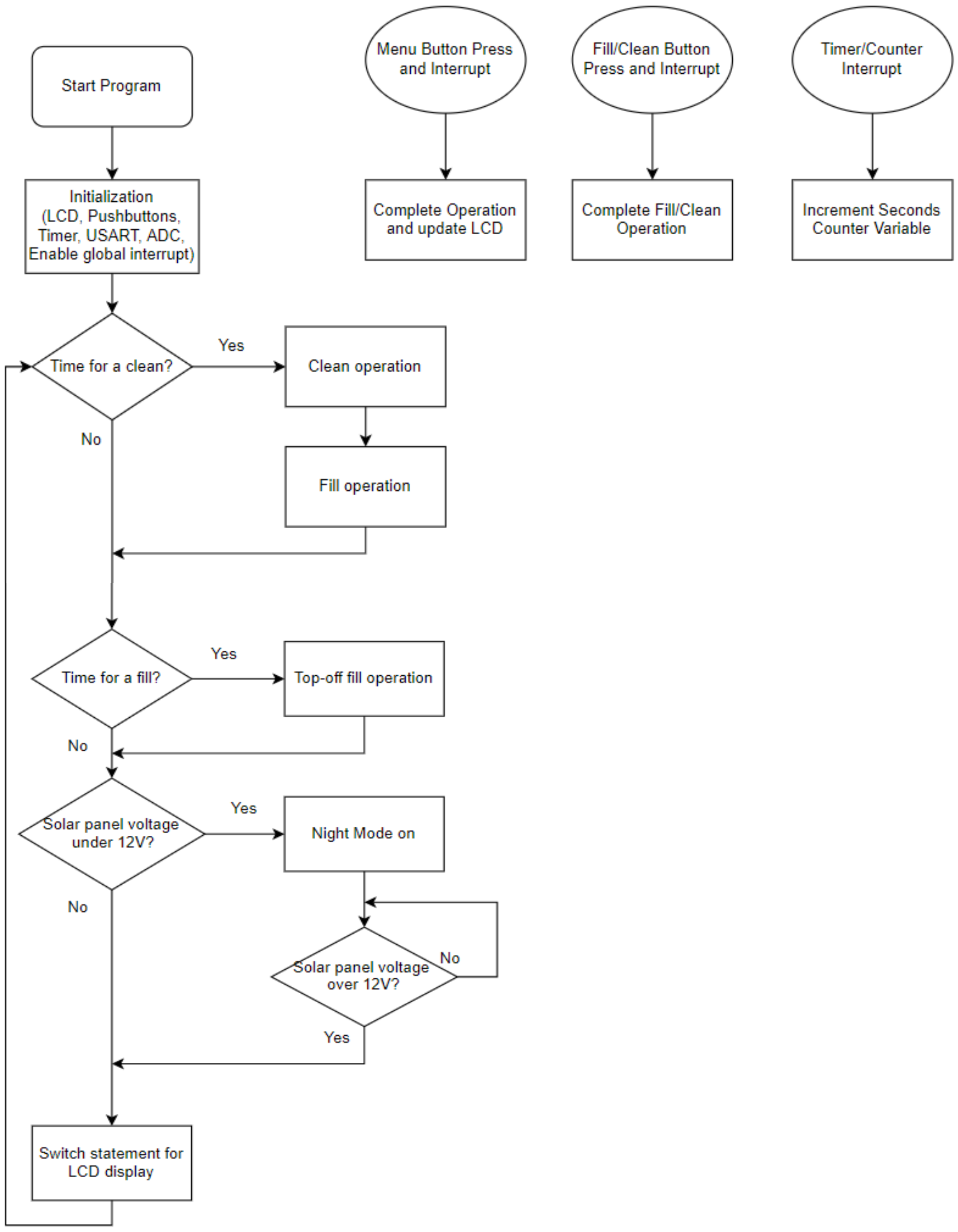

Start Program
Initialization (LCD, Pushbuttons, Timer, USART, ADC, Enable global interrupt)
Time for a clean?
Yes
Clean operation
Fill operation
No
Time for a fill?
Yes
Top-off fill operation
No
Solar panel voltage under 12V?
Yes
Night Mode on
Solar panel voltage over 12V?
No
Yes
No
Switch statement for LCD display
Menu Button Press and Interrupt
Complete Operation and update LCD
Fill/Clean Button Press and Interrupt
Complete Fill/Clean Operation
Timer/Counter Interrupt
Increment Seconds Counter Variable

Figure 4: A flowchart of the current software used in the design

Timing of System

The timing of the system is carried out by TCA0 of the AVR128DB48. The TCA is a 16-bit timer/counter that can be used for program execution timing, frequency and waveform generation, and command execution. For the purposes of this project, TCA0 keeps track of time in seconds. A counter value is compared to a period value, and when the counter value reaches the period value, an interrupt request occurs. Because TCA0 keeps track of time in seconds, the interrupt needs to take place every second. For TCA0 to interrupt every second, the period value must be set to allow this. The formula below was used to calculate the period value:

$$TCA_{period} = \frac{t * CLK\_MAIN}{TCA_{prescaler}} - 1$$

where t = desired time between each interrupt, CLK_MAIN = main clock frequency of microcontroller, $TCA_{prescaler}$ = prescaler value of TCA. Inputting the desired values, the equation becomes:

$$TCA_{period} = \frac{1(s) * 4MHz}{256} - 1$$

$$TCA_{period} = 15,624$$

It takes a second for the counter value to get to this period value of 15,624. In the interrupt service routine (ISR) of TCA0, a 16-bit variable named second_counter is incremented. This variable keeps tracks of the seconds that have passed. When second_counter reaches the time a fill cycle is expected to occur, the system starts a fill and second_counter keeps incrementing afterwards. In the case of a clean cycle, after the clean cycle is done, second_counter resets to zero.

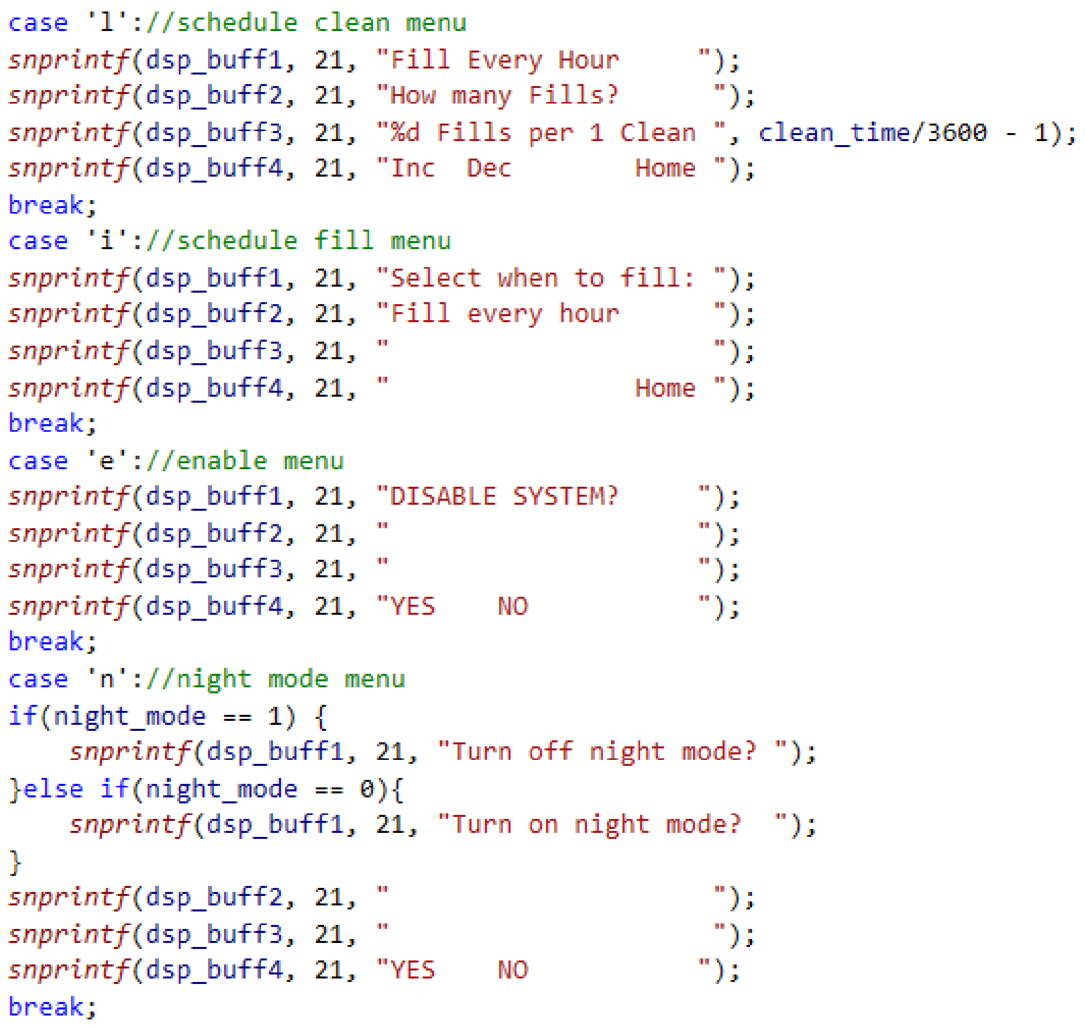
LCD and Button Interface

The LCD interface utilizes a finite-state machine in which each state corresponds to a menu option for the LCD. There are currently 12 states, or screens, as shown in Table 2.

mode	Description
d	Displays “System Disabled”
h	Home menu that shows the when the last cycle was as well as when the next two will occur
l	Schedule clean menu that allows user to increment or decrement the number of cleans per fill
i	Schedule fill menu that allows user to set the duration of the top-off fill cycle in seconds
e	Disable system menu. Asks whether to disable system.
n	Enable night mode menu. Asks whether to turn on/off night mode
f	Fill menu that displays how much time is left in the fill cycle
c	Clean menu that displays how much time is left in the clean cycle
m	Mode menu that shows the IP address of the WiFi webpage and allows the user to disable the system or night mode
a	Diagnostics menu that displays the voltage from the SSRs
d	Displays “Night Mode Enabled”

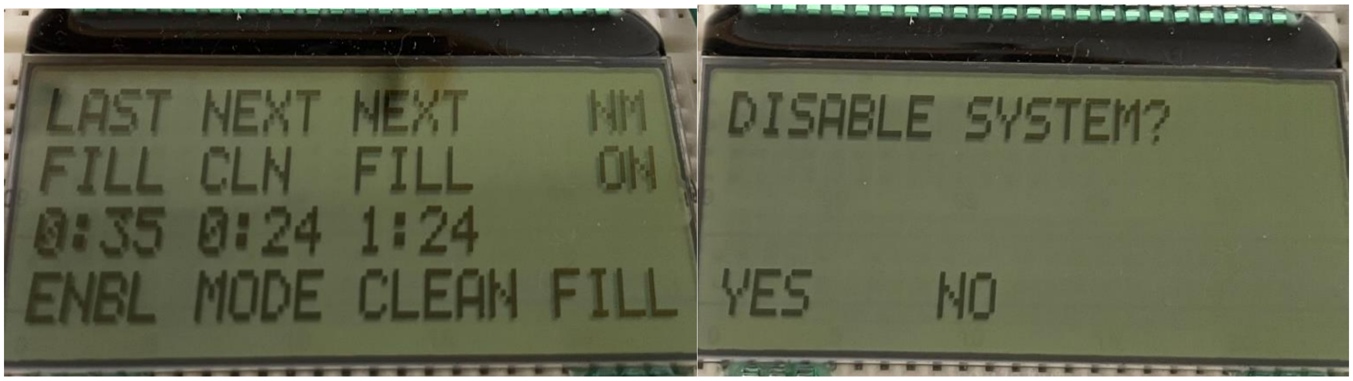
Table 2: Letter and their corresponding screens.

The system stores the letter of the screen in a char variable called “mode”. In the main loop, there is a switch statement that is determined by mode. The switch statement shows the corresponding LCD screen. **Error! Reference source not found.** shows a snippet of the code.



```
case 'l'://schedule clean menu
snprintf(dsp_buff1, 21, "Fill Every Hour     ");
snprintf(dsp_buff2, 21, "How many Fills?      ");
snprintf(dsp_buff3, 21, "%d Fills per 1 Clean ", clean_time/3600 - 1);
snprintf(dsp_buff4, 21, "Inc  Dec         Home ");
break;
case 'i'://schedule fill menu
snprintf(dsp_buff1, 21, "Select when to fill: ");
snprintf(dsp_buff2, 21, "Fill every hour      ");
snprintf(dsp_buff3, 21, "                     ");
snprintf(dsp_buff4, 21, "                Home ");
break;
case 'e'://enable menu
snprintf(dsp_buff1, 21, "DISABLE SYSTEM?     ");
snprintf(dsp_buff2, 21, "                    ");
snprintf(dsp_buff3, 21, "                    ");
snprintf(dsp_buff4, 21, "YES    NO           ");
break;
case 'n'://night mode menu
if(night_mode == 1) {
    snprintf(dsp_buff1, 21, "Turn off night mode? ");
}else if(night_mode == 0){
    snprintf(dsp_buff1, 21, "Turn on night mode?  ");
}
snprintf(dsp_buff2, 21, "                     ");
snprintf(dsp_buff3, 21, "                     ");
snprintf(dsp_buff4, 21, "YES    NO           ");
break;
```

Figure 5: Part of the code for the main loop switch statement.

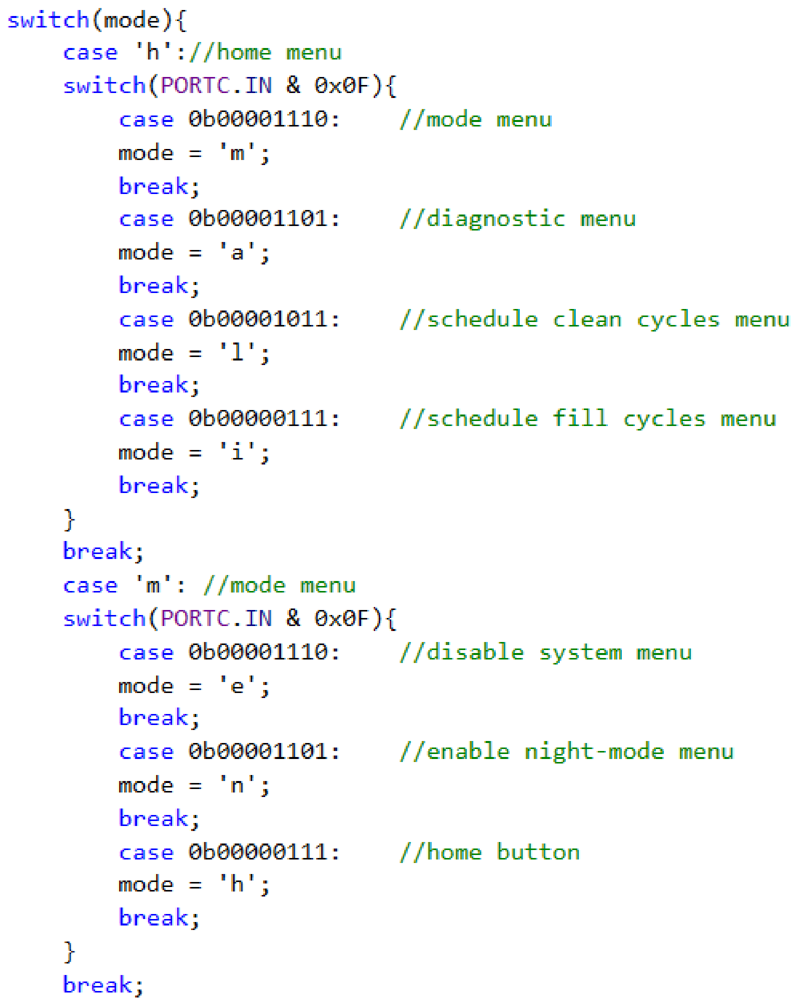


```
LAST NEXT NEXT    NM
FILL CLN  FILL    ON
0:35 0:24 1:24
ENBL MODE CLEAN FILL
```

```
DISABLE SYSTEM?


YES    NO
```

Figure 6: LCD screens showing the home screen and disable system screen.

For the pushbutton interface, the program uses interrupts to recognize a press. The LCD has four buttons associated with it that are connected to pins 0-3 of port C of the AVR128DB48. When a button is pressed, the program goes to the ISR for port C. Like the LCD interface, in the ISR, there is a switch statement that changes the function of the pushbuttons based on what letter mode currently holds. In Figure 7, the code shows that if mode is currently ‘h’, the four buttons will act as the mode menu button, diagnostics button, schedule clean button, and schedule fill button. If the user were to press the button associated with the mode menu, the switch statement would change to case ‘m’; this would then change the function of the pushbuttons to the disable system menu, enable night mode menu, and the home menu.

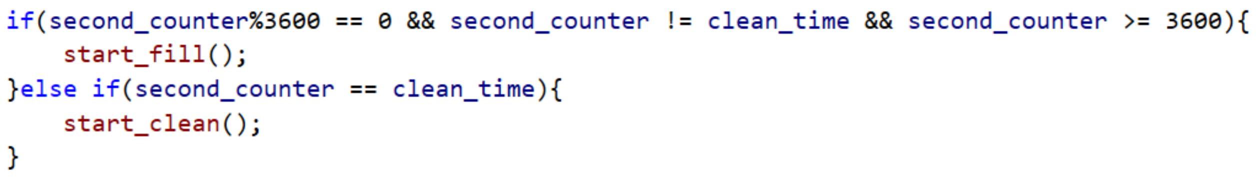


```
switch(mode){
    case 'h'://home menu
    switch(PORTC.IN & 0x0F){
        case 0b00001110:    //mode menu
        mode = 'm';
        break;
        case 0b00001101:    //diagnostic menu
        mode = 'a';
        break;
        case 0b00001011:    //schedule clean cycles menu
        mode = 'l';
        break;
        case 0b00000111:    //schedule fill cycles menu
        mode = 'i';
        break;
    }
    break;
    case 'm': //mode menu
    switch(PORTC.IN & 0x0F){
        case 0b00001110:    //disable system menu
        mode = 'e';
        break;
        case 0b00001101:    //enable night-mode menu
        mode = 'n';
        break;
        case 0b00000111:    //home button
        mode = 'h';
        break;
    }
    break;
```

Figure 7: Part of the ISR for port C

Clean and Fill Cycles

The clean and fill cycles take place when second_counter reaches their scheduled times. For the clean cycle, this is user defined and defaults to 10800 seconds, or every three hours. The fill cycle stays at 3600 seconds, or every hour.



```
if(second_counter%3600 == 0 && second_counter != clean_time && second_counter >= 3600){
    start_fill();
}else if(second_counter == clean_time){
    start_clean();
}
```

Figure 8: Code that schedules a fill or clean cycle.

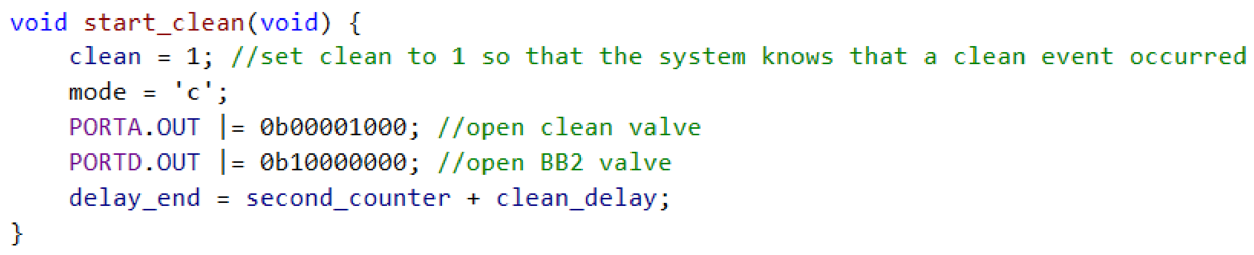
In Figure 8, a fill cycle is initiated when second_counter is divisible by 3600, or after an hour. It also makes sure that when it is time for a clean cycle, a fill cycle does not also happen. For example, when second_counter is at 10800, this number is divisible by 3600 and it is equal to the scheduled clean time (clean_time in the code). Instead of both cycles taking place, only the clean cycle should be activated. The last requirement for a fill cycle is that second_counter should not be less than 3600 because 0 mod 3600 is zero, but the fill cycle should not occur when second_counter is zero. The only requirement for a clean cycle is that second_counter is equal to the scheduled clean time.

Phase	Clean Valve	Fill Valve	BB2 Valve	Duration
Clean Phase 1	ON	OFF	ON	15
Clean Phase 2	ON	OFF	OFF	30
Fill	OFF	ON	ON	45
Standby	OFF	OFF	OFF	-----

Table 3: Duration of clean cycle and fill cycle as well as which valves are on during the cycles.

Clean Cycle

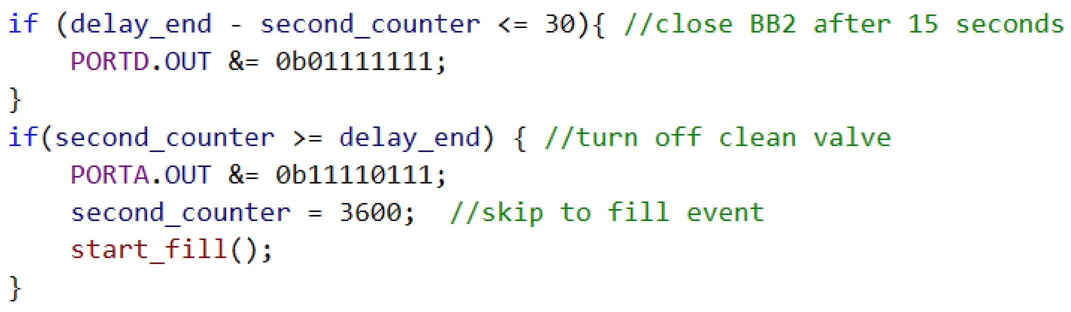
To start a clean cycle, the SSRs for the clean valve and BB2 valve are enabled, allowing water to flow to the top and bottom bird baths. There is a variable, clean, that is set to 1. This variable is used to tell the system whether a clean event has occurred before a fill event because there are two types of fill cycles. This will be explained further later in the report. The variable mode is set to ‘c’ so that the LCD displays the clean menu. There is also a variable, delay_end, that holds the value of second_counter at the beginning of the clean cycle added with the length of the cycle (45 seconds). This tells the system when the cycle will end.



```
void start_clean(void) {
    clean = 1; //set clean to 1 so that the system knows that a clean event occurred
    mode = 'c';
    PORTA.OUT |= 0b00001000; //open clean valve
    PORTD.OUT |= 0b10000000; //open BB2 valve
    delay_end = second_counter + clean_delay;
}
```

Figure 9: Code to start a clean cycle.

The bird bath system consists of two bird baths. There is a top bird bath that is closer to the water supply and a bottom bird bath on a lower level and further away. As a result, the bottom bird bath does not get much water during a clean cycle. To combat this problem, a third valve is used to block water from reaching the top bird bath, which then allows all the water to flow to the bottom one. The following code shows this as well as the end of the clean cycle.



```
if (delay_end - second_counter <= 30){ //close BB2 after 15 seconds
    PORTD.OUT &= 0b01111111;
}
if(second_counter >= delay_end) { //turn off clean valve
    PORTA.OUT &= 0b11110111;
    second_counter = 3600;  //skip to fill event
    start_fill();
}
```

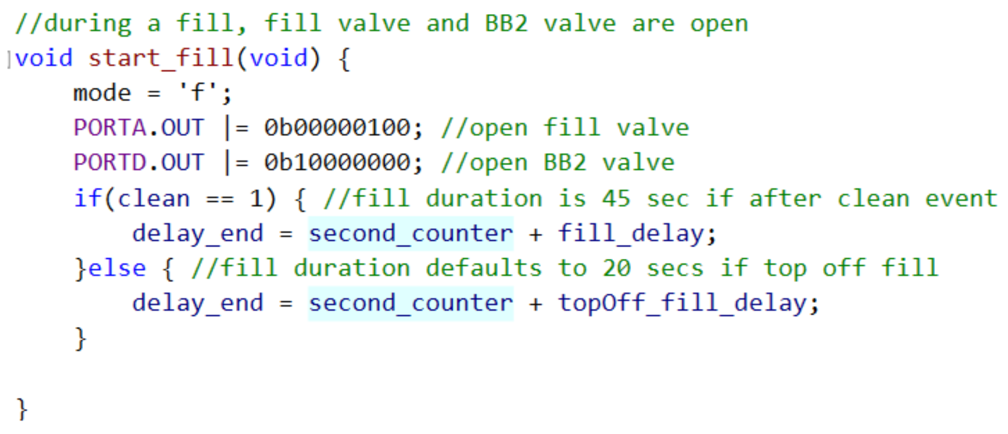
Figure 10: Code showing phase 1 and phase 2 of clean cycle.

In Table 3, it is shown that during the first phase of the clean cycle, the clean valve and the BB2 valve are open for 15 seconds. In the second phase of the clean cycle, BB2 closes to allow all the water to flow to the bottom bird bath and lasts 30 seconds. In the code snippet from figure 6, the BB2 valve is closed when second_counter subtracted from delay_end is less than 30. As an example, if second_counter is 10800 and delay_end is 10845 at the beginning of the clean cycle, the first if statement would be false since 45 is greater than 30. Second_counter would eventually increment to 10815, in which case the first if statement would be true since 30 is equal to 30. From this point onwards the BB2 valve is always off. The second if statement checks to see if second counter is greater than or equal to delay_end. If it is true, the clean cycle is done, the clean valve can be closed, and the fill cycle follows.

<u>Fill Cycle</u>

There are two types of fill cycles: the fill cycle after a clean cycle and the fill cycle that occurs every hour (top-off fill). The fill after a clean takes 45 seconds and can only be changed in the actual code. The top-off fill defaults to 20 seconds at start up, but the time can be increased or decreased in the schedule fill menu.

The code for the fill cycle is the same regardless of the type. To start a cycle, the SSRs for the fill valve and the BB2 value are enabled, allowing water to go to both bird baths. The mode variable changed to ‘f’ so that the LCD displays the fill menu. For the system to know when to end the cycle, delay_end is used here as well. If there was a clean cycle before the fill, the clean variable would have been set to one. Depending on whether clean is a one, delay_end is either 45 seconds greater than the value of second_counter at the beginning of the function or 20 seconds greater.



```
//during a fill, fill valve and BB2 valve are open
void start_fill(void) {
    mode = 'f';
    PORTA.OUT |= 0b00000100; //open fill valve
    PORTD.OUT |= 0b10000000; //open BB2 valve
    if(clean == 1) { //fill duration is 45 sec if after clean event
        delay_end = second_counter + fill_delay;
    }else { //fill duration defaults to 20 secs if top off fill
        delay_end = second_counter + topOff_fill_delay;
    }

}
```

Figure 11: Code to start a fill cycle.

To stop a fill cycle, both the fill valve and the BB2 valve are closed. If the system was disabled before the cycle, the LCD returns to the disabled menu. If not, the LCD returns to the home menu. For a top-off fill, second_counter continues to increment. For the fill cycle after a clean, second_counter resets to zero. The clean variable is also changed to zero so that the next time a fill cycle occurs, the system will know that it is a top-off fill.

Wi-Fi Hardware and Software Design Discussion

Wi-Fi Overview and Design Motivation

The Wi-Fi module is a key part of the upgraded design, it allows the user to access the BBAC interface from a device as long as it's on the wi-fi network and can run a web browser. This opens up the ability to manage the interface from far away like inside a house on a rainy or hot day. It also opens up the ability to integrate the BBAC with a smart home or other internet connected environment in the future with upgrades to the system.

Wi-Fi Module Choice

The Wi-Fi module being used is the ESP8266 in the form of the ESP-01s, shown in Figure 12. The main reasons it was chosen was size, availability, price, and ease of development. This module has a small form factor and can be integrated into the PCB with a socket, meaning that if there ever is an issue or it needs to be reprogrammed, it can easily be removed and replaced. These modules also are very common and have a low price, costing only about $2-3 each, meaning it is not costly or difficult to replace them if necessary. It also is easy to develop for, with wide community support and extensive wireless libraries available. [4]

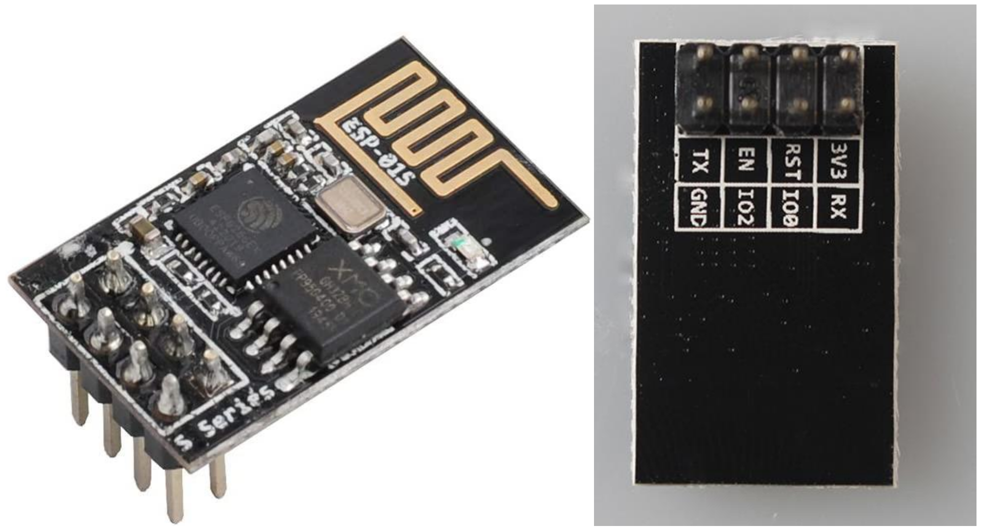


Figure 12: The ESP-01s module and its corresponding pinout

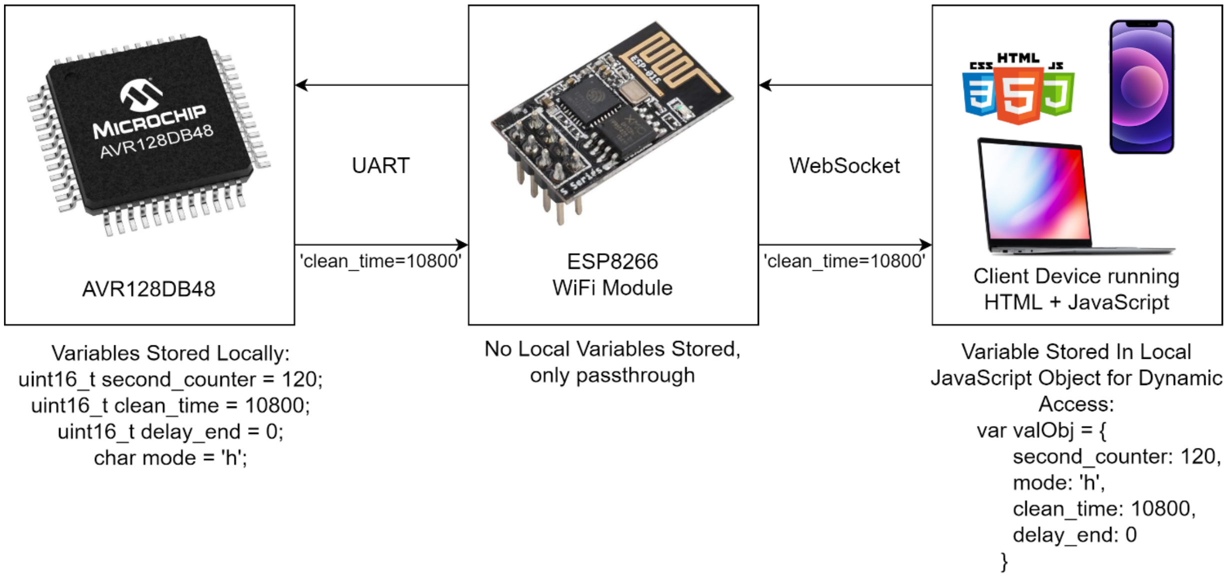


Figure 13: A diagram showing the interaction between the 3 devices, including communication protocols, examples of local variables, and example strings being passed through

<u>Communication between ESP-01s and AVR128</u>

UART was chosen for easy 2 wire communication and easy plain text debugging with a PC and serial debugger. The UART connection is made between the AVR device and the ESP device and allows them to communicate at high speeds with the baud rate of 115200. The ESP device passes the strings received over UART to the web client using WebSocket, where it is then processed by the webpage using JavaScript.

The UART setup code is shown below. The setup code initializes the USART0 device on the AVR microcontroller with a baud rate of 115200, and also makes stdout forward to the USART_stream. This means that standard printf commands can be used to send strings to USART. Figure 15 shows an example of the setup code used for initalizing USART0 with stdout to the USART_stream.

```
#define USART0_BAUD_RATE(BAUD_RATE) ((float)(3333333 * 64 / (16 *(float)BAUD_RATE)) + 0.5)

FILE USART_stream = FDEV_SETUP_STREAM(USART0_printChar, NULL, _FDEV_SETUP_WRITE);

void USART0_init(void)
{
		PORTA.DIR &= ~PIN1_bm;
		PORTA.DIR |= PIN0_bm;

		USART0.BAUD = (uint16_t)USART0_BAUD_RATE(115200);
		USART0.CTRLB |= USART_TXEN_bm;
		//USART0.CTRLB |= USART_RXEN_bm; add receive function later

		stdout = &USART_stream;

}
```

Figure 14: Example code for the USART0 initialization

The transmission currently occurs within the TCA0 interrupt. Because the baud rate is very high, it takes very little time so does not affect the interrupt’s timing much. Strings are formatted in a way so that it has variable name on the left of the equal sign, then the value on the right side of the equal sign, with no spaces. This is so that the variable can be easily parsed by the Javascript function running on the webpage.

```
ISR(TCA0_OVF_vect) {
		second_counter += 0x01; // increment seconds counter

		printf("second_counter=%d\n", second_counter);
		printf("clean_time=%d\n", clean_time);
		printf("delay_end=%d\n", delay_end);
		printf("mode=%c\n", mode);
		TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm; //clear interrupt flags
}
```

Figure 15: Example code for USART transmission inside the TCA0 interrupt

In the future, there is the possibility for more variables to be transmitted, and these could be added anywhere in the program with a simple printf statement. For example, if the transmission of variables takes up too much time in the timer counter interrupt, we can move everything but the second_counter update outside the function to update only when a sync function is run or one of the variables changes. For example, a sync would be run when the ESP device initially connects over USART by sending a sync command, then it would be run periodically. This method of syncing uses less time on the USART because variables are only sent when updated or the periodic sync is run. This, however, has more of a chance to desync if somehow the USART command is corrupted or interrupted compared to sending variables within the interrupt.

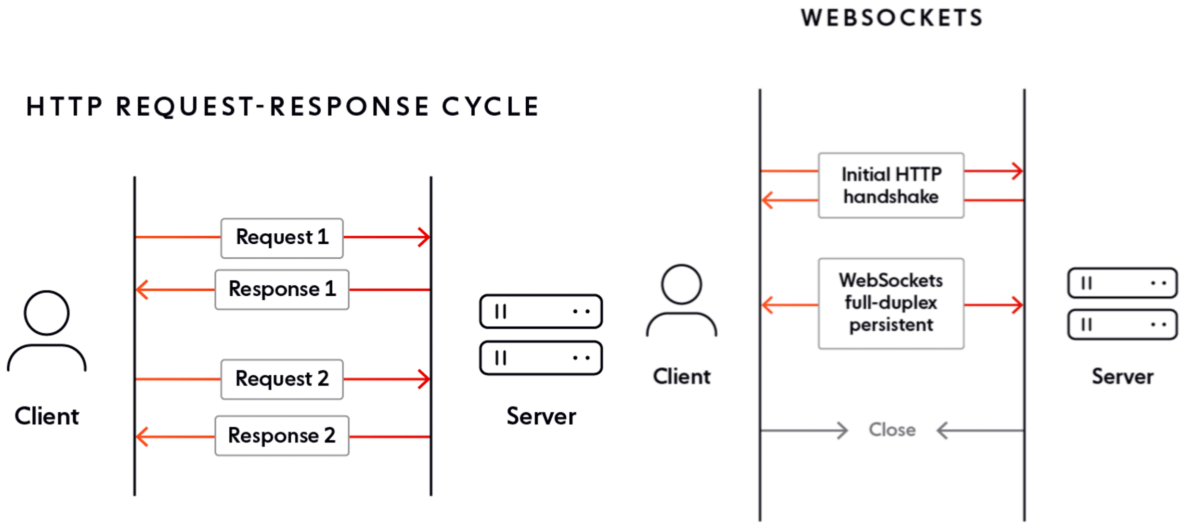


Figure 16: Diagrams illustrating the difference between HTTP requests with polling and WebSocket (asynchronous) [9]

Communication between ESP server and Web Client

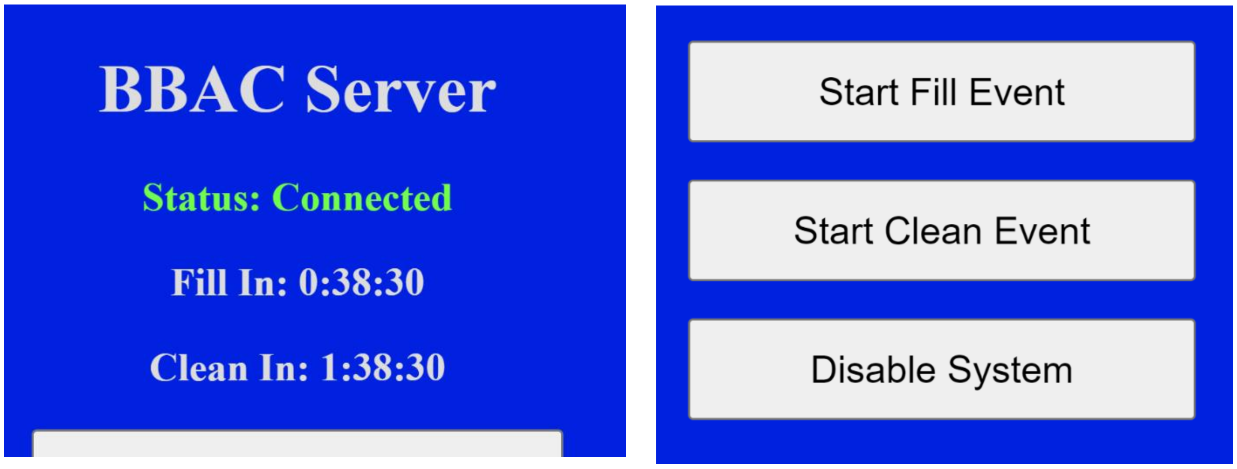


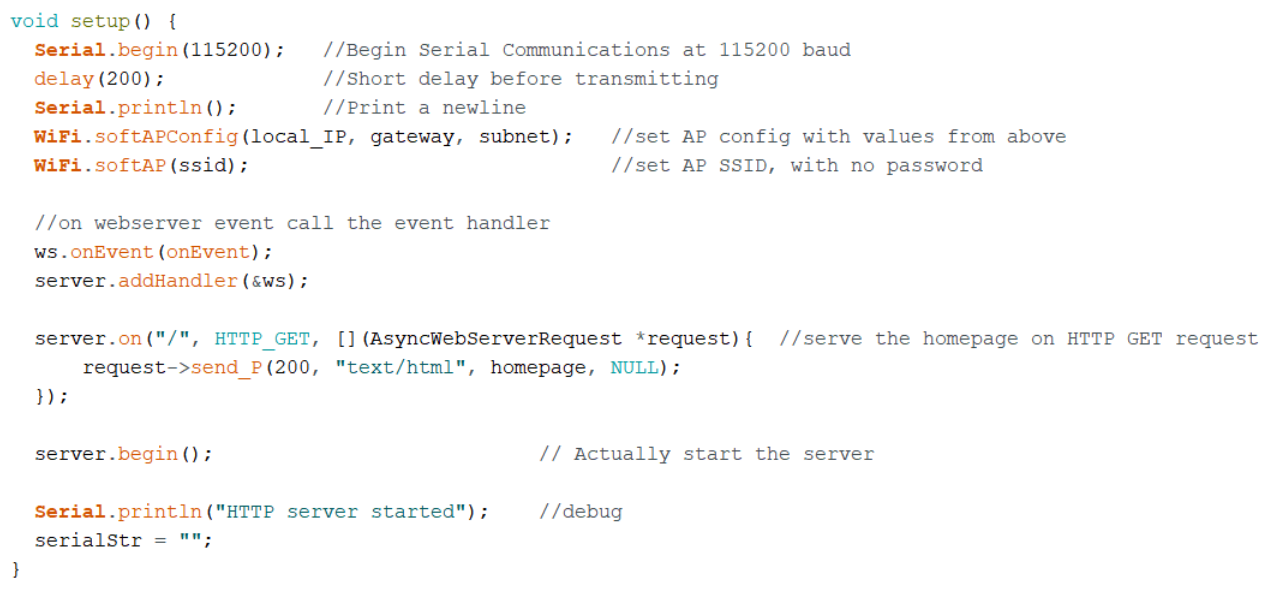
Figure 17: A screenshot of the web interface

WebSocket is a technology that allows for asynchronous receive and transmit between a server and a client. In this case, the server is the ESP8266 device, and the client is the web browser visiting the website. After a website is opened, WebSocket opens a stream of communication between the server and the client using JavaScript. When receiving a message, JavaScript code is run to parse the message and update variables that might have been changed. Similarly, the ESP server runs code when it receives a message, passing the data back to the AVR microcontroller to be processed.

This technology updates on receive or transmit, with no need for polling every x milliseconds like a traditional AJAX implementation with HTTP GET requests. This means instantaneous updates and less stress on the server, which is important with our small, low power ESP device hosting the website. Figure 16 illustrates the difference between HTTP AJAX and WebSocket. With HTTP, individual requests are sent and received, so polling is necessary to update at a regular rate. With WebSocket, a connection is opened, then communication can occur anytime with no need for polling. [9]

The setup code for the ESP8266 web server is shown below. This code includes the start of serial communications at a 115200 baud rate. Next, it sets up Wi-Fi communication in host mode (‘softAP’), meaning it broadcasts a signal as an unsecured access point. The ESP also has the ability to connect to a network in station (‘STA’) mode where it is a device on an existing access point’s network. This can be used to connect to a local network where it can be accessed by any other device on the network.

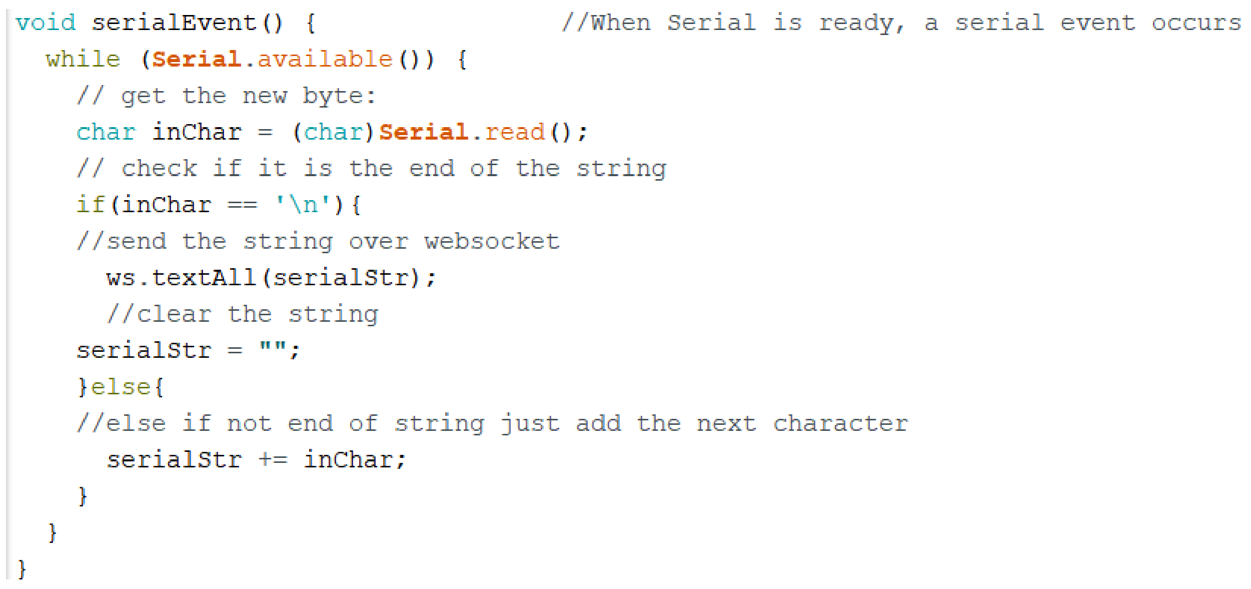
After setup of the Wi-Fi network, the server is initialized, it passes a handler function (onEvent) that happens when something is received over WebSocket. This function is described in more detail in the ‘Sending Commands through WebSocket’ section. The next function is to handle HTTP requests. This happens when the user looks up the ESP’s webpage on a web browser. The device serves an HTTP page with all visual elements and JavaScript code that is to be run by the browser. This HTTP page is stored in a large string variable called homepage for simplicity. Finally, with all configuration steps completed, the server is started with the server.begin() function.



```
void setup() {
  Serial.begin(115200);   //Begin Serial Communications at 115200 baud
  delay(200);             //Short delay before transmitting
  Serial.println();       //Print a newline
  WiFi.softAPConfig(local_IP, gateway, subnet);   //set AP config with values from above
  WiFi.softAP(ssid);                              //set AP SSID, with no password

  //on webserver event call the event handler
  ws.onEvent(onEvent);
  server.addHandler(&ws);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){  //serve the homepage on HTTP GET request
      request->send_P(200, "text/html", homepage, NULL);
  });

  server.begin();                          // Actually start the server

  Serial.println("HTTP server started");   //debug
  serialStr = "";
}
```

Figure 18: Setup code for the ESP8266 webserver

Handling of incoming serial data and the transmission of the data is shown below. When a character is received over serial (USART), it is appended to a string called serialStr. Once a newline character is received, the whole string is sent over WebSocket (ws.textAll) to be processed by the JavaScript on the webpage.



```
void serialEvent() {                //When Serial is ready, a serial event occurs
  while (Serial.available()) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // check if it is the end of the string
    if(inChar == '\n'){
    //send the string over websocket
      ws.textAll(serialStr);
      //clear the string
    serialStr = "";
    }else{
    //else if not end of string just add the next character
      serialStr += inChar;
    }
  }
}
```

Figure 19: Serial handling code on ESP8266

WebSocket Receive and Processing on the Webpage

After the data is sent over WebSocket, it is handled by JavaScript in the browser. Figure 20 shows the processing of the data with JavaScript. valObj is defined when the website opens, an object is used because it allows for dynamic naming. This means that strings can be used to create new variables or to access existing ones within an object. Because of this, the WebSocket data can easily be split and immediately used for accessing a variable.

For example, an input string might be “second_counter=15”, which is split on the equal sign into two strings: “second_counter” and “15”. The variable name is then used to access it in the object (valObj[“second_counter”]) and set it equal to “15”. The “15” is automatically converted to a number when it is used in a mathematical operation. This conversion is known as dynamic typing and is one of the reasons that JavaScript is able to easily process these formatted strings. This splitting code runs every time a string is received by the webpage over WebSocket (webSocket.onmessage).

```
var valObj = {
    second_counter: 0,
    mode: 'f',
    clean_time: 0,
    delay_end: 0
  }

webSocket.onmessage = function(event) {
    var data = event.data;
    splitDat = data.split('=');
    valObj[splitDat[0]] = splitDat[1];
```

Figure 20: Data Processing with JavaScript

Sending Commands through WebSocket and Processing on the AVR

Commands are able to be sent through WebSocket to, which then are passed through UART to the AVR. This can be used for triggering events like clean or fill, or for changing numerical values by passing variables. To accomplish this, websocket has an ‘onEvent’ function that sends the command string to the AVR processor. This onEvent function simply runs the handleIncomingData when it receives data. The AVR processes when the whole command is received, indicated by a newline character at the end of the command. The strcmp function is used to check the string against possible matches and run the appropriate command.

```
void handlingIncomingData(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode ==
WS_TEXT) {
    data[len] = 0;
    Serial.printf((char*)data);
  }
}
```

Figure 21: The handlingIncomingData function used for sending commands from WebSocket to serial (UART)

Breadboard Testing and PCB Design

Demonstration Board Construction

All components and modules were wired onto one breadboard for testing and demonstrations. Namely: the board consisted of the power supply unit and fuses, the socketed AVR128DB28, the JTAG connector for flashing the microcontroller, the solid-state relays, the LCD screen and Softkeys, the ESP8266 Wi-Fi module and bread board connector, a DIP switch and voltage divider circuit network for analog diagnostics and the solenoids attached to the SSRs.

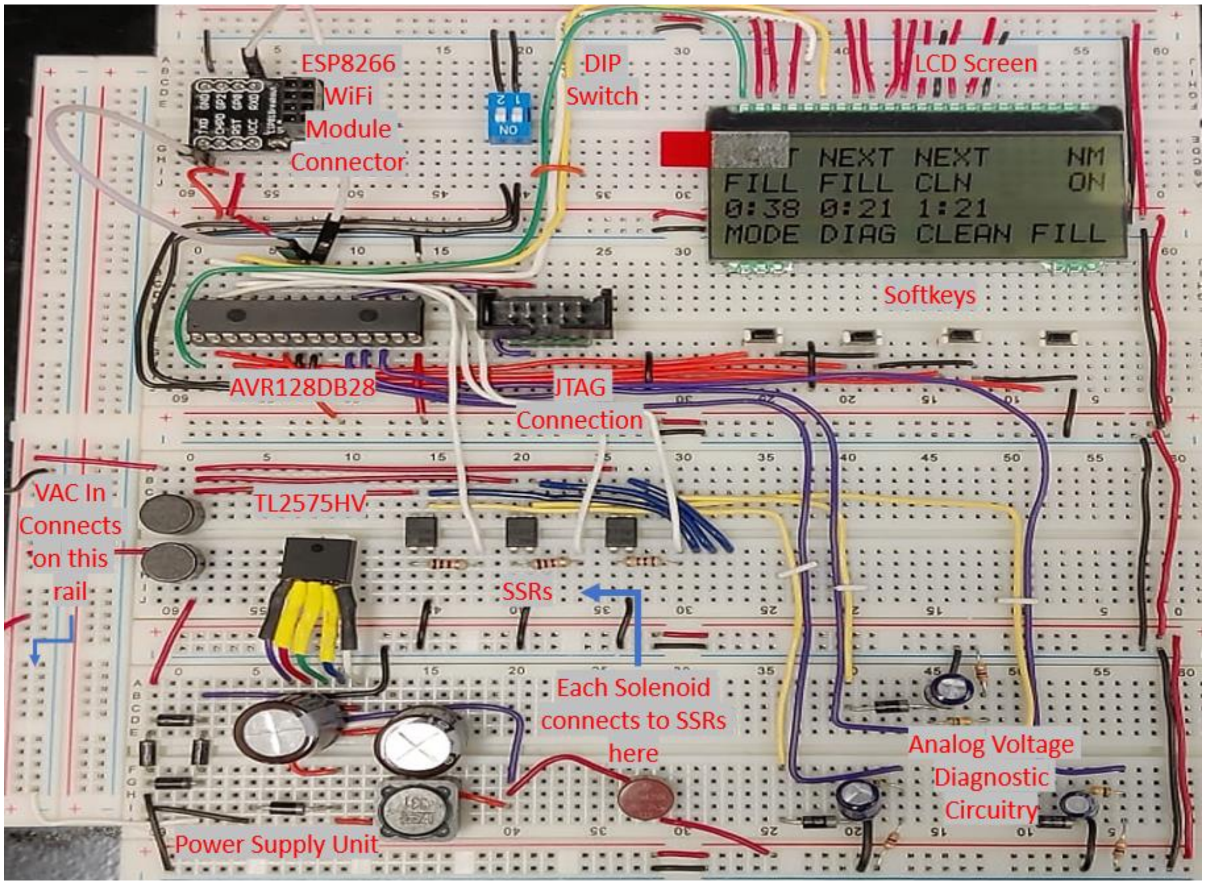


Figure 22: An overhead view of the demonstration board layout

PCB Design

After this breadboard design was completed and verified working, the next step was to design the PCB layout. The design was created on Fusion360. Based on the schematic of the entire system, all the parts were transferred to the PCB side. The focus of the PCB is the LCD, so it was placed in the middle, with the pushbuttons that control it placed under. The headers that connect the solenoids, external pushbutton, and AC power supply were placed on the sides for easy connections with the external components. The resistors and capacitors were placed behind the LCD to save space, as the LCD is raised on the PCB using female headers. All the other components were placed around the LCD. Trace routing on the PCB was done after testing on the demonstration board.

Boards were ordered from JLCPCB, who was chosen based on their affordable pricing, high quality board printing, and quick turnaround time. The PCB was ordered in the beginning of April and assembling the board was done around the middle of the month.

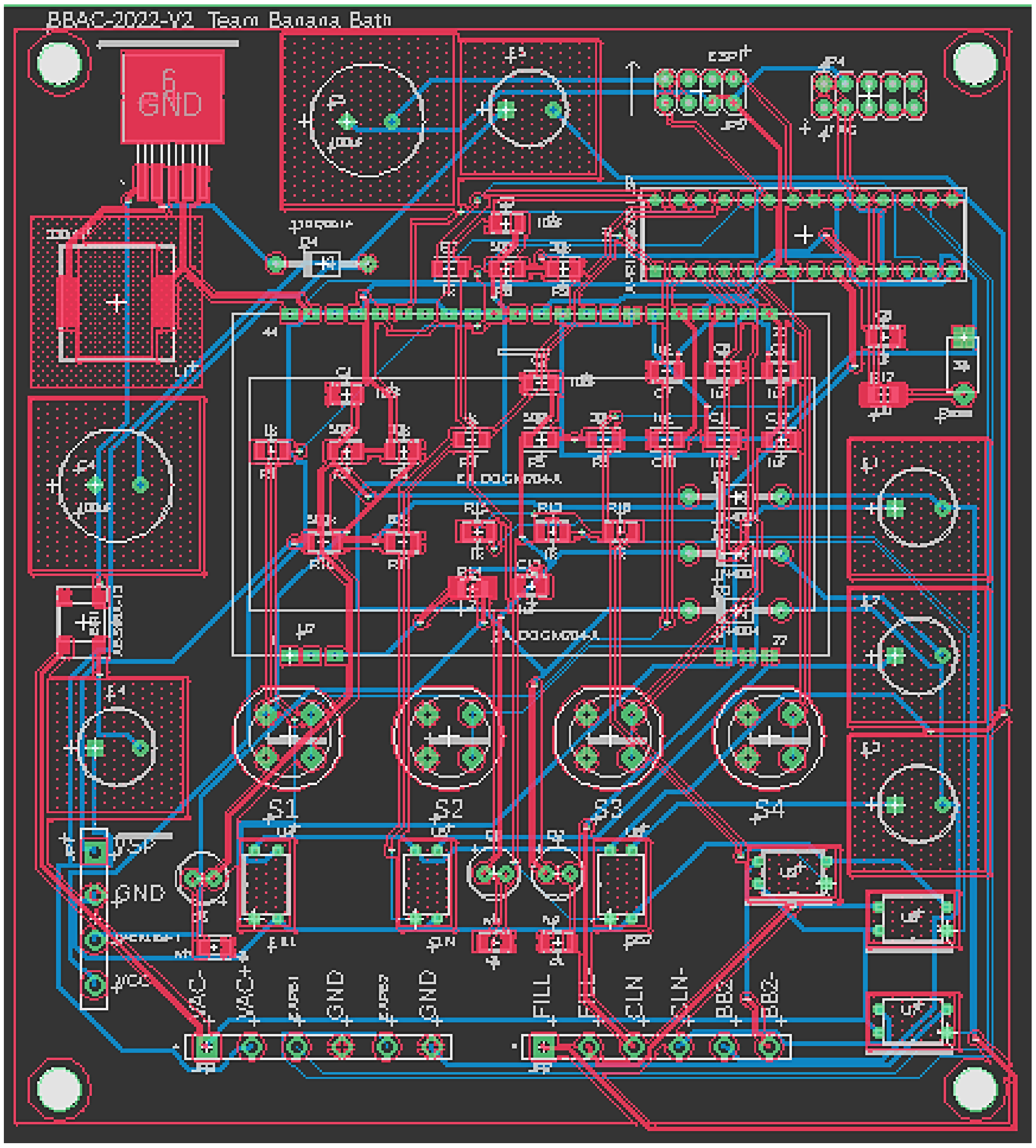


Figure 23: The Final PCB Design, V2

Final Results

The board was assembled near the end of April, and it was installed on site on April 30th. The board has been running since with a few minor software issues that were updated like too long clean and fill timing. As the board continues to be used in the field, we observe how the user (Scott) interacts with it and make updates to our program based on his feedback.

Some issues arose with version 1 of the board, such as the ESP8266 mounting point being too small. A custom mount was made and the ESP8266 was successfully mounted on the V1 board. There were also some poor connection issues with the LCD from the low-quality female connectors that were used, but these were fixed after putting some solder on the ends of the pins of LCD so that there was not such a loose connection. The pinout for the JTAG connectors, while functional, was incorrect as well since it was an 8-pin connector rather than a 10-pin connector. There were also silk screen inconsistencies, such as D4 being backwards, which affected the assembly of the board since the power supply did not initially work. These issues will be addressed in the BBAC-V2 board that was ordered about a week after installing the first one. There also will be a dedicated negative connection for the negative SSRs that was not present on the V1 board. This will allow diagnostics to work properly on the new V2 board.

Overall, the project was very successful; the features that were promised were implemented and delivered on time. Although the V1 board is not perfect, it is still fully functional with all core features except diagnostics, which will be fixed in V2 of the board. Wi-Fi, the LCD, buttons, solenoid triggering, and the power supply are all working properly on the current installation.

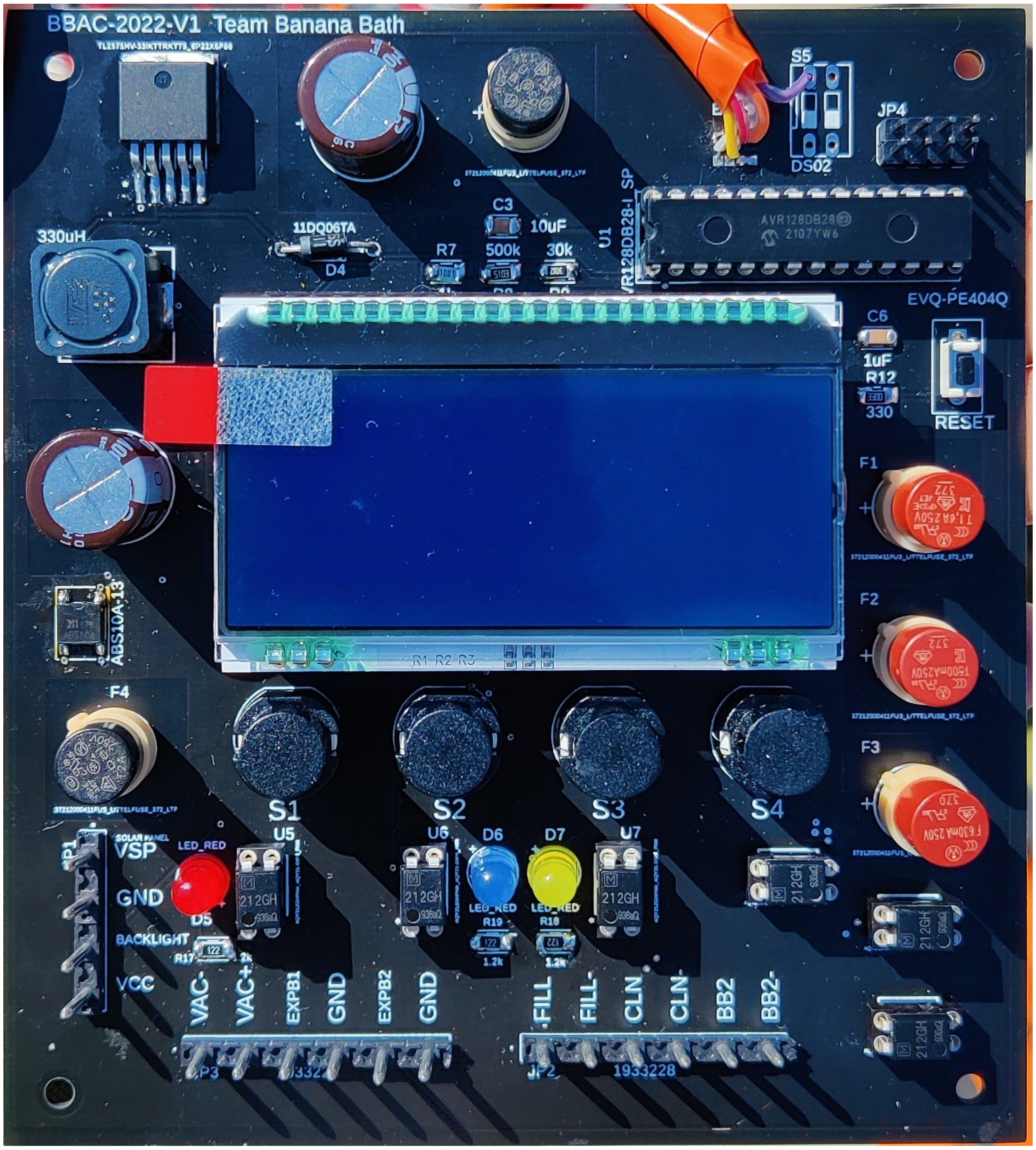


Figure 24: The V1 PCB fully built before installation

Discussion

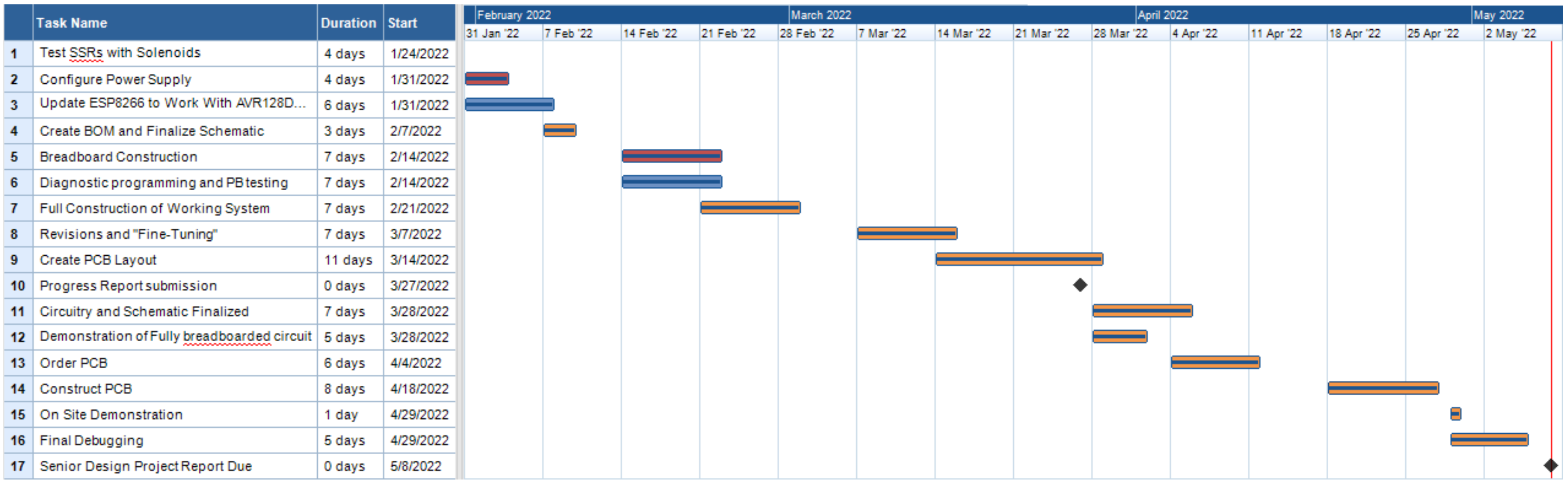
During the year, we learned a good deal of new information in all aspects of the project. New components that we had never used in a project was introduced to us such as solid-state relays, LCDs, and DC-DC switching regulators. Coding the AVR128DB48 in C was also new to most of us. By working on this project we were able to gain experience and knowledge.

Ethics are important to consider in any design project, and for this project we consider the final product to be ethical and even helpful for the environment. The main ethical issues involving this project is how it interacts with the local environment and if that is a positive or negative thing. The project is a bird bath, so it does give a good environment for local birds to drink and wash, this is a positive for the environment around the project. Protecting the control module in a weather-proof box, and protecting the system from power spikes with fuses, also ethically protects the user and the environment from potential harm and malfunction. There is, however, a possible negative ethical concern with water usage, as the project does use more water than a manually filled bird bath. There is even the concern of electrical utility charges increasing since the system uses AC power and is always plugged in and on. This, however, is not too much an issue as we included features like night mode that prevent waste of water and power by turning off the system at night.

The way we interacted with others during this project was also, we believe, ethical. Throughout the project, we gave truthful and accurate progress update with our advisor, and we always strived to be transparent and professional in our communication. This professionalism extended to our work together as a team, and it helped keep the team on good terms with no major conflicts. We learned a lot about interacting professionally with teammates and with authority figures throughout the project.

There were also many issues that came about in both the hardware and software aspects of the project, whether it be components that did not perform as expected or the program. Some of these challenges include the problems with power supply selection, issues with the V1 PCB, and problems getting software to work. However, the team was able to work together, along with Scott, to solve these problems. We also became better at communicating with each other when there were disagreements about design choices. Throughout the course of this project, not only did we learn more about the technical aspects, but we also learned about how to work cohesively as a group. Going forward, this knowledge can be very beneficial in future project or job roles that may come, as working together is an important part of life in general.

Appendix A: Completed Gannt Chart



	Task Name	Duration	Start
1	Test SSRs with Solenoids	4 days	1/24/2022
2	Configure Power Supply	4 days	1/31/2022
3	Update ESP8266 to Work With AVR128D...	6 days	1/31/2022
4	Create BOM and Finalize Schematic	3 days	2/7/2022
5	Breadboard Construction	7 days	2/14/2022
6	Diagnostic programming and PB testing	7 days	2/14/2022
7	Full Construction of Working System	7 days	2/21/2022
8	Revisions and "Fine-Tuning"	7 days	3/7/2022
9	Create PCB Layout	11 days	3/14/2022
10	Progress Report submission	0 days	3/27/2022
11	Circuitry and Schematic Finalized	7 days	3/28/2022
12	Demonstration of Fully breadboarded circuit	5 days	3/28/2022
13	Order PCB	6 days	4/4/2022
14	Construct PCB	8 days	4/18/2022
15	On Site Demonstration	1 day	4/29/2022
16	Final Debugging	5 days	4/29/2022
17	Senior Design Project Report Due	0 days	5/8/2022

Chart Key

Task done by Brandon Banarsi	🟥
Task done by Brandon Guzy	🟦
Task done by Vanessa Li	🟩
Task done by Entire Team	🟧
Completed tasks	—
Milestone	◆

Appendix B: Record of Team Meetings

Date	Description	Participants
2/7/22	First meeting of the semester, discussion of PCB design, part choices	Scott Tierno, Brandon Guzy, Brandon Banarsi, Vanessa Li
2/17/22	Meeting to drop off breadboard parts, discuss design	Scott Tierno, Brandon Guzy, Vanessa Li
2/28/22	Demonstrate breadboard and PCB design progress	Scott Tierno, Brandon Guzy, Brandon Banarsi, Vanessa Li
3/11/22	Meeting to work on finishing PCB and Breadboard design	Scott Tierno, Brandon Banarsi, Vanessa Li
3/21/22	Discussion of power supply issues with ESP chip	Scott Tierno, Brandon Banarsi, Vanessa Li
3/28/22	Work on power supply issue, power supply issue resolved and system working fully	Brandon Guzy, Brandon Banarsi, Vanessa Li
4/1/22	Demonstration of power supply working, PCB discussion	Scott Tierno, Brandon Guzy, Brandon Banarsi
4/3/22	Meeting to finish Presentation	Brandon Guzy, Brandon Banarsi, Vanessa Li
4/11/22	Meeting about POSTs and diagnostics	Scott Tierno, Brandon Guzy, Vanessa Li
4/15/22	Meeting about SSR solution to diagnostic problem, backlight	Scott Tierno, Brandon Guzy, Brandon Banarsi, Vanessa Li
4/22/22	Meeting to Build PCB board	Scott Tierno, Brandon Guzy,

		Vanessa Li
4/25/22	Meeting to Finish PCB board and work on extender	Scott Tierno, Brandon Guzy, Brandon Banarsi, Vanessa Li
4/29/22	Meeting to install Wi-Fi mounting extender and fully finish PCB	Brandon Guzy, Brandon Banarsi, Vanessa Li
4/30/22	Meeting to demo and install PCB on site	Scott Tierno, Brandon Guzy, Brandon Banarsi, Vanessa Li

Appendix C: Works Cited

[1]

Panasonic, “AQY212 Datasheet.” Mar. 2021. [Online]. Available: https://www3.panasonic.biz/ac/e_download/control/relay/photomos/catalog/semi_eng_gu1a_aqy21_g.pdf

[2]

Microchip, “AVR128DB28-32-48-64-DataSheet.” 2020. Accessed: Dec. 02, 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/AVR128DB28-32-48-64-DataSheet-DS40002247A.pdf

[3]

Display Visions, “EA DOGM204-A Datasheet.” Jul. 2021. Accessed: Dec. 02, 2021. [Online]. Available: https://www.lcd-module.de/fileadmin/eng/pdf/doma/dogm204e.pdf

[4]

AI-Thinker team, “ESP-01 WiFi Module Datasheet.” 2015. Accessed: Dec. 02, 2021. [Online]. Available: https://www.microchip.ua/wireless/esp01.pdf

[5]

Texas Instruments, “LM340, LM340A and LM7805 Datasheet.” Sep. 2016. Accessed: Dec. 02, 2021. [Online]. Available: https://www.ti.com/lit/ds/symlink/lm340.pdf

[6]

Maxim Integrated, “MAX5035 Datasheet.” May 27, 2011. Accessed: Dec. 02, 2021. [Online]. Available: https://datasheets.maximintegrated.com/en/ds/MAX5035.pdf

[7]

Micrel, Inc., “MIC2937 Datasheet.” May 2006. Accessed: Dec. 02, 2021. [Online]. Available: https://ww1.microchip.com/downloads/en/DeviceDoc/mic2937.pdf

[8]

Texas Instruments, “TL2575, TL2575HV. Datasheet.” Jan. 2006. Accessed: May 08, 2022. [Online]. Available: https://www.ti.com/lit/ds/symlink/tl2575-05.pdf

[9]

K. Kilbride-Singh, “WebSockets vs Long Polling,” *Ably Blog: Data in Motion*, Oct. 27, 2021. https://ghost.ably.com/blog/websockets-vs-long-polling/ (accessed Dec. 02, 2021).

[10]

V. R. Schmalhofer, “Why Birds Matter,” *Center For Earth and Environmental Science - Indiana University*, Feb. 11, 2018. https://cees.iupui.edu/blog/why-birds-matter

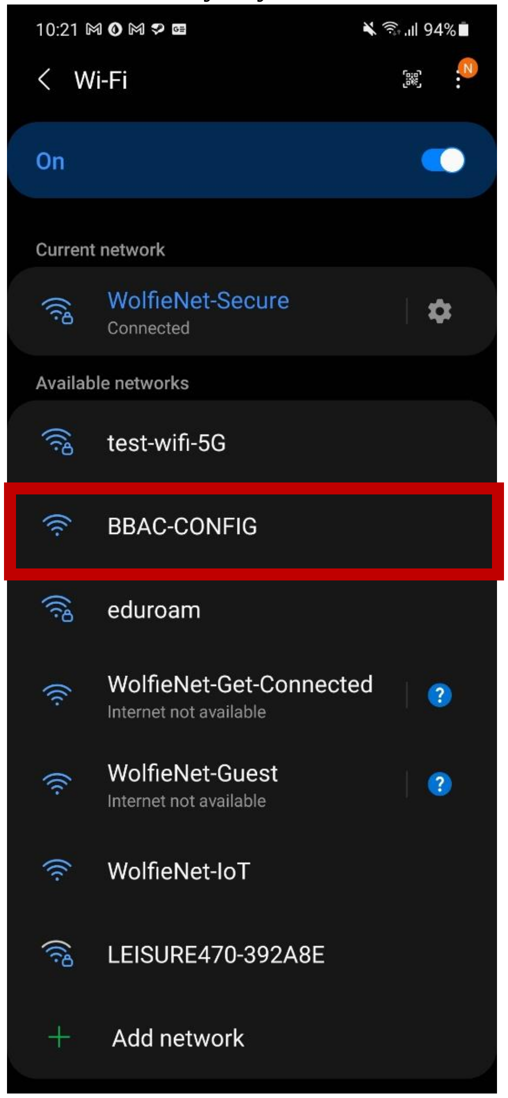
Appendix D: Wi-Fi Guide

This guide covers the setup of the wi-fi access point. After, it describes the wi-fi hardware used and software used for development.

Software Usage Guide

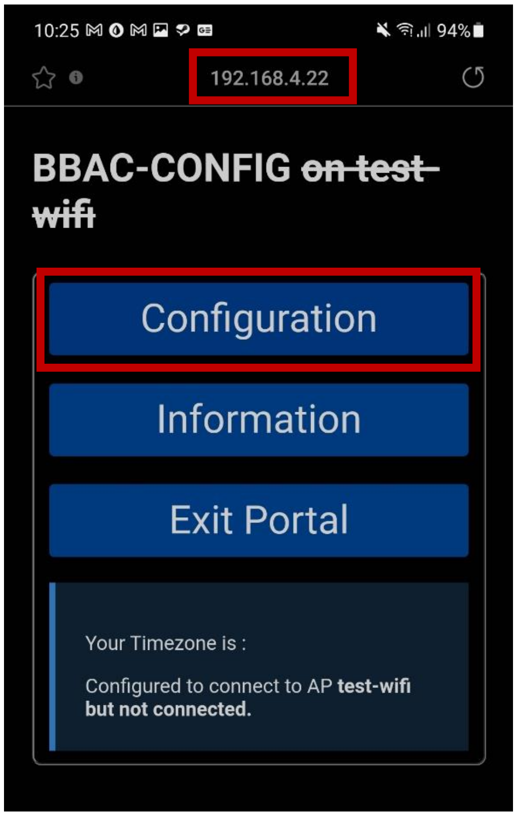
Connecting to an Access Point

1. When the ESP8266 first starts, it will attempt to connect to the saved access point. If it connects, the following login steps will be skipped, and it will launch immediately into the BBAC UI.
2. If the ESP8266 does not successfully reconnect to the saved access point, it will enter config mode after a minute. Connect to the BBAC-Config network. If warned about no internet connectivity, agree to connect anyways.



3. Go to IP address 192.168.4.22 on your web browser.

4. Tap Configuration. Alternatively, for information on the device like MAC address, tap ‘Information’

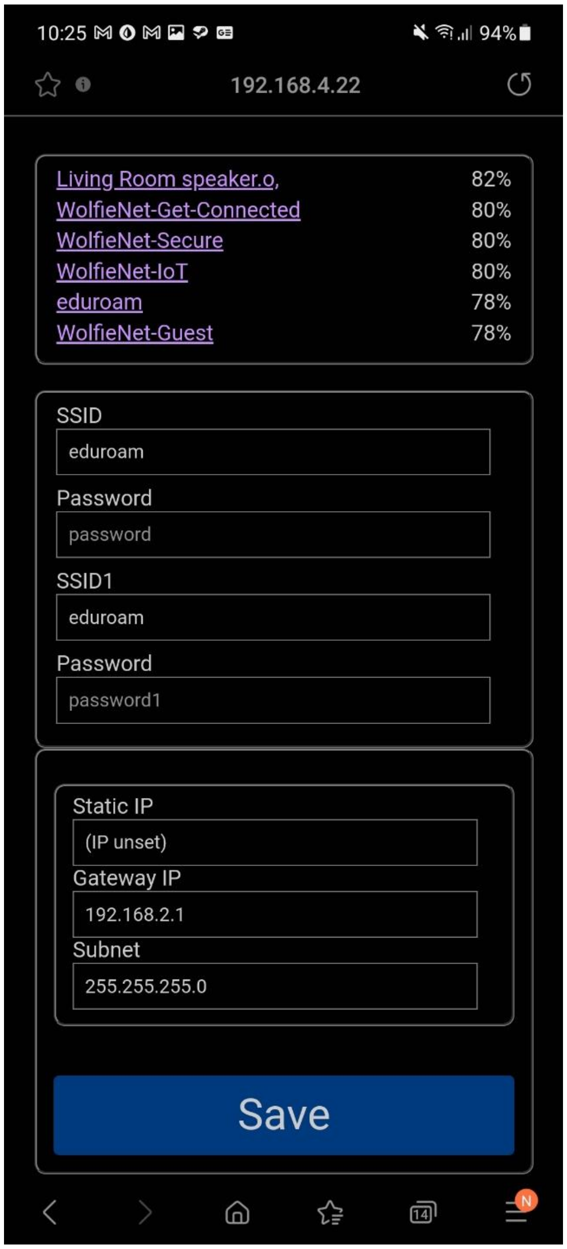


BBAC-CONFIG ~~on test-wifi~~

Your Timezone is :

Configured to connect to AP **test-wifi but not connected.**

5. Tap a wi-fi access point to copy its SSID. Enter the password in the ‘password’ field


10:25 94%

192.168.4.22

Living Room speaker.o,	82%
WolfieNet-Get-Connected	80%
WolfieNet-Secure	80%
WolfieNet-IoT	80%
eduroam	78%
WolfieNet-Guest	78%

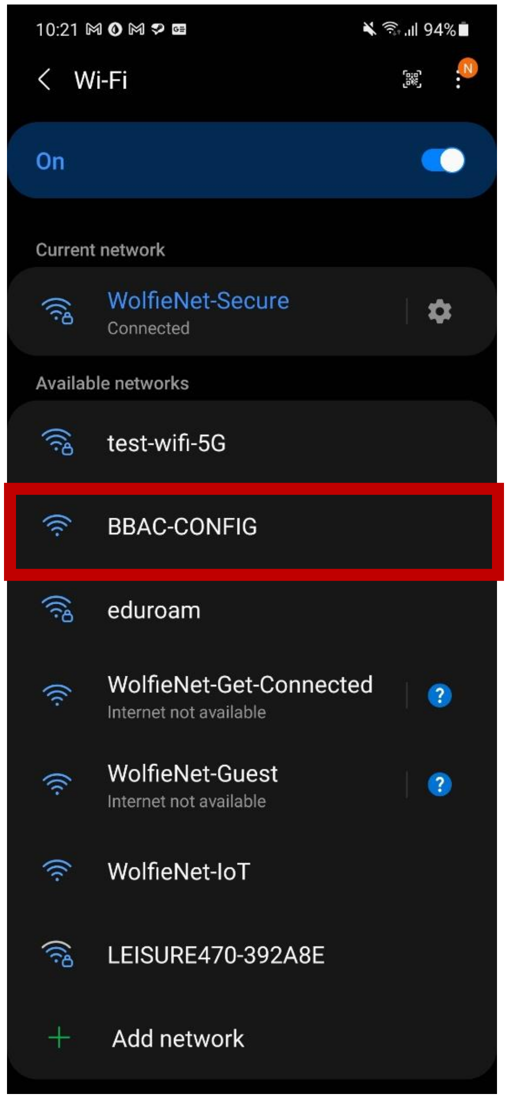
SSID: eduroam
Password: password
SSID1: eduroam
Password: password1

Static IP: (IP unset)
Gateway IP: 192.168.2.1
Subnet: 255.255.255.0

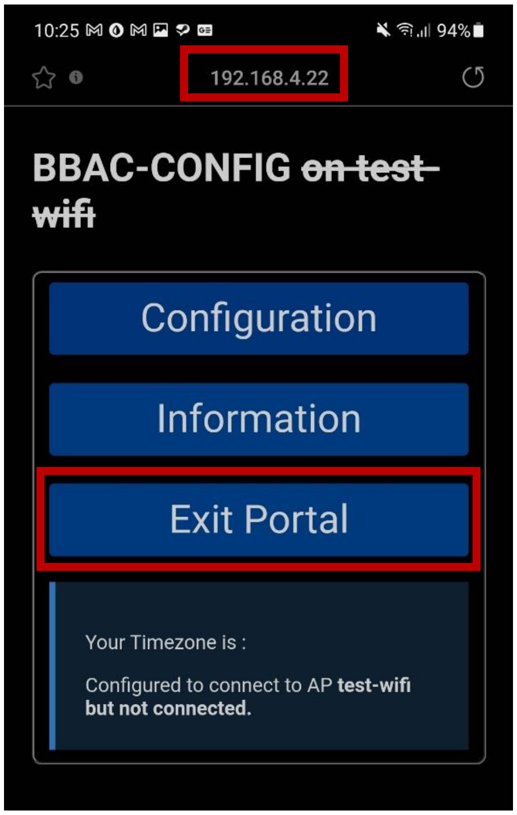
6. Tap save and wait a minute for the device to connect to the access point
7. Check the LCD under ‘mode’ to see the IP address on the local network the ESP has been assigned. Navigate to this IP address in a browser to access the BBAC interface

<u>Self-Hosting</u>

1. When the ESP8266 first starts, it will attempt to connect to the saved access point. If it connects, the following login steps will be skipped, and it will launch immediately into the BBAC UI.
2. If the ESP8266 does not successfully reconnect to the saved access point, it will enter config mode after a minute. Connect to the BBAC-Config network. If warned about no internet connectivity, agree to connect anyways.



3. Go to IP address 192.168.4.22 on your web browser.
4. Tap Exit Portal

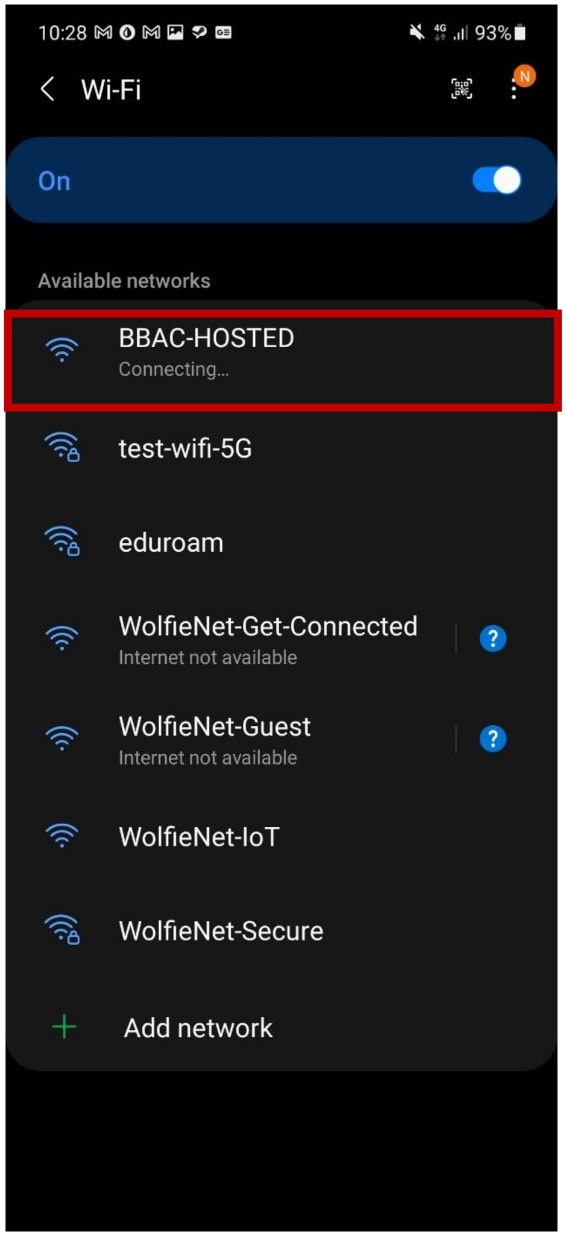


BBAC-CONFIG ~~on test-wifi~~

Your Timezone is :

Configured to connect to AP **test-wifi but not connected.**

5. The BBAC will enter self hosting mode. Connect to the BBAC-HOSTED network. Agree to connect even if warned about internet connectivity



6. Go to IP address 192.168.4.22 to access the BBAC interface
7. To Exit hosted mode, reset the BBAC by unplugging and re-plugging it in or powering it down and up again.

Software Installation and Hardware Guide

Hardware Used

The ESP-01s was used for this project, it can be found on websites like eBay and Amazon for affordable prices (usually around $3-5 per chip). The flasher used was a DIYmall usb flasher with CH340C chip. The flasher usually work without driver installation, but sometimes CH340C drivers need to be installed, install packages can be found at this link or through google searches.

Software IDE Used-Arduino

The Arduino IDE was chosen for its robust community support for the ESP8266, while official tools are available, they do not have as many features and do not have as many community libraries. Using community libraries and software made the development of this portion of the project much easier. A full IDE setup guide is in the 'helpful resources section' and a PDF is included with the Arduino code.

Libraries Used

Some community libraries were used for this project because it would take months of effort to implement these features on my own. The libraries are the ESP8266 arduino core, ESP Async Wifi Manger, ESP Async web Server, and ESP Async TCP. The links are included in the 'helpful resources' section below, and they are also included as zip files with the Arduino code in case the github repositories are removed.

Web Development

Web development was done using pure html and javascript in a single file. This was done so that the homepage HTML could be included in a single .h file, so no complex file systems needed to be used with the ESP8266. The ESP8266 has a file system mode so it can host more complex webpages with multiple html, css, js, and other files like images. This mode was deemed too complicated to setup and replicate so the single .h file was chosen.

Web development can be done in a separate .html file then copied over to the homepage.h file, this allows the page to be previewed on a local browser before copying changes over to the .h file and flashed.

Helpful Resources

ESP8266 arduino core documentation

https://arduino-esp8266.readthedocs.io/en/latest/index.html

Arduino IDE setup guide

https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/

WebSocket Guide

https://randomnerdtutorials.com/esp8266-nodemcu-websocket-server-arduino/

ESPAsyncTCP

https://github.com/me-no-dev/ESPAsyncTCP

ESPAsyncWebServer

https://github.com/me-no-dev/ESPAsyncWebServer

ESP 8266 Arduino core

https://github.com/esp8266/Arduino

ESP Async WiFi Manager

https://github.com/khoih-prog/ESPAsync_WiFiManager

Appendix E: Source Files

AVR Code

Main File

```c
#define F_CPU 4000000UL

#include <avr/io.h>
#include <string.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include <stdio.h>
#include "DOG204_LCD.h"
#include "USART_config.h"
#include "ADC_diagnostic.h"

void start_fill(void);
void start_clean(void);
void cancel_fill(void);
void cancel_clean(void);

char IPAdd[21];
uint16_t second_counter = 0;
uint16_t clean_time = 10800;
uint16_t delay_end = 0;
uint8_t fill_delay = 45;           //duration of fill event
uint8_t topOff_fill_delay = 20;    //duration of top off fill
const uint8_t clean_delay = 45;    //duration of clean event
int night_mode = 0;         //1 means that night mode is on and 0 mean that it is off
int clean = 0;              //1 means that a clean event was right before the fill event
int disabled = 0;

//MODE List:
// h = home menu
// f = filling menu
// c = cleaning menu
// d = system disable menu
// l = schedule clean menu
// i = schedule fill menu
// n = enable night-mode menu
// g = night-mode menu
// s = sleep menu
// e = disabled system menu
// a = diagnostic menu
// m = mode menu
char mode = 'h';

//***************************************************************************
//
// Function Name        : "PORTB ISR"
// Date                 : 9/20/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      : Push Button
// Author               : Brandon Guzy & Vanessa Li
// DESCRIPTION
// Interrupt service routine for port B, handles button presses in different
// modes. For each mode, buttons are assigned different functions like switching
// to another screen or triggering actions like a fill/clean
//
// Warnings               : none
// Restrictions           : none
// Algorithms             : none
// References             : none
//
// Revision History       : v1.1 Added additional mode options
//
//***************************************************************************
ISR(PORTC_PORT_vect){

       switch(mode){
              case 'h'://home menu
              switch(PORTC.IN & 0x0F){
                     case 0b00001110:     //mode menu
                     mode = 'm';
                     break;
                     case 0b00001101:     //diagnostic menu
                     mode = 'a';
                     break;
                     case 0b00001011:     //schedule clean cycles menu
                     mode = 'l';
                     break;
                     case 0b00000111:     //schedule fill cycles menu
                     mode = 'i';
                     break;
              }
              break;
              case 'm': //mode menu
              switch(PORTC.IN & 0x0F){
                     case 0b00001110:     //disable system menu
                     mode = 'e';
                     break;
                     case 0b00001101:     //enable night-mode menu
                     mode = 'n';
                     break;
                     case 0b00000111:     //home button
                     mode = 'h';
                     break;
              }
              break;
              case 'e'://disable system menu
              switch(PORTC.IN & 0x0F){
                     case 0b00001110:     //yes to disable mode
                     mode = 'd';
                     break;
                     case 0b00001101:     //no to disable mode
                     mode = 'm';
                     break;
              }
              break;
              case 'd'://menu that shows "SYSTEM DISABLED"
              switch(PORTC.IN & 0x0F){
                     case 0b00000111:     //turn off disabled mode
```

```
		mode = 'h';
		break;
	}
	break;
	case 'n'://enable night-mode menu
	if(night_mode == 1) {
		switch(PORTC.IN & 0x0F){
			case 0b00001110:	//yes to turn off night mode
			night_mode = 0;
			mode = 'h';
			break;
			case 0b00001101:	//no to turn off night mode
			mode = 'm';
			break;
		}
	}else {
		switch(PORTC.IN & 0x0F){
			case 0b00001110:	//yes to turn on night mode
			night_mode = 1;
			mode = 'h';
			break;
			case 0b00001101:	//no to turn on night mode
			mode = 'm';
			break;
		}
	}
	break;
	case 'l'://schedule number of fills per clean
	switch(PORTC.IN & 0x0F){
		case 0b00001110:		//increment fills per clean
		if(clean_time < 63000)	//cannot overflow, max 18 hour clean time
			clean_time += 3600;
		break;
		case 0b00001101:		//decrement fills per clean
		if(clean_time > 10800)	// cannot go under 2 fills per clean
			clean_time -= 3600;
		break;
		case 0b00001011:		//start clean cycle
		start_clean();
		break;
		case 0b00000111:		//home button
		if(second_counter > clean_time) {
			clean_time = clean_time + 3600;
		}
		mode = 'h';
		break;
	}
	break;
	case 'i'://select duration of top-off fill
	switch(PORTC.IN & 0x0F){
		case 0b00001110:		//increment by 1 second
			topOff_fill_delay += 1;
		break;
		case 0b00001101:		//decrement by 1 second
		if(topOff_fill_delay > 1)	//cannot go below 1 second
			topOff_fill_delay -= 1;
		break;
		case 0b00001011:		//home button
```

```
                            mode = 'h';
                     break;
                     case 0b00000111:            //start fill cycle
                            start_fill();
                     break;
              }
              break;
              case 'f'://filling menu
                     cancel_fill();
              break;
              case 'c'://cleaning menu
                     cancel_clean();
              break;
              case 'a': //diagnostics menu
              switch(PORTC.IN & 0x0F){
                     case 0b00000111:     //home button
                     mode = 'h';
                     break;
              }
              break;
              case 'g': //night-mode menu
              switch(PORTC.IN & 0x0F){
                     case 0b00000111:   //disable night mode
                     night_mode = 0;
                     mode = 'h';
                     break;
              }
              break;
       }
       _delay_ms(250);
       PORTC_INTFLAGS = 0xFF;      //clear interrupt flags
}

//***************************************************************************
//
// Function Name        : "execute_USART_command"
// Date                 : 12/5/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; none
// Author               : Brandon Guzy
// DESCRIPTION
// This takes a string as an input and executes commands based on what
// the string is. Meant to be used in conjunction with the USART recieve
// interrupt
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void execute_USART_command(char myCommand[]){
       if(!strcmp(myCommand, "fill")){
              start_fill();
       }else if(!strcmp(myCommand, "clean")){
```

```c
                start_clean();
        }else if(!strcmp(myCommand, "disable")){
                mode = 'd';
        }else if(!strncmp(myCommand, "IP:", 3)){
                strcpy(IPAdd, myCommand);
        }else if(!strcmp(myCommand, "cancel")){
                if (mode == 'c') {
                        cancel_clean();
                }else if(mode == 'd'){
                        mode = 'h';             //turn off disabled mode
                }else if (mode == 'f') {
                        cancel_fill();
                }
        }
}

//***************************************************************************
//
// Function Name        : "PORTF ISR"
// Date                 : 12/07/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      : Push Button
// Author               : Vanessa Li
// DESCRIPTION
// Interrupt service routine for port F, handles button presses for the
// external pushbuttons. PORT0 performs a clean and PORT1 performs a fill.
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     :
//
//**************************************************************************
ISR(PORTF_PORT_vect){
        switch(PORTF.IN & 0b00000011){
                        case 0b00000001:        //external clean
                                if(mode == 'c') {
                                //if currently cleaning, cancel if button is pressed
                                        cancel_clean();
                                }else if(mode == 'f') {
                                //if currently filling, cancel fill if button is pressed
                                        cancel_fill();
                                }else {
                                        start_clean();
                                }
                        break;
                        case 0b00000010:        //external fill
                                if(mode == 'f') {
                                        cancel_fill();
                                }else {
                                        start_fill();
                                }
                        break;
                }
        _delay_ms(250);
```

```
       PORTF_INTFLAGS = 0xFF;      //clear interrupt flags
}

//***************************************************************************
//
// Function Name        : "TCA0_init"
// Date                 : 10/16/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; none
// Author               : Vanessa Li
// DESCRIPTION
// This program initializes the TCA0
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void TCA0_init(void)
{
 /* enable overflow interrupt */
 TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
 /* set Normal mode */
 TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc;
 /* disable event counting */
 TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTAEI_bm);
 /* set the period */
 TCA0.SINGLE.PER = 15624;
 /* set clock source (sys_clk/256) */
 TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV256_gc
                    | TCA_SINGLE_ENABLE_bm; /* start timer */
}
//***************************************************************************
//
// Function Name        : "TCA0 ISR"
// Date                 : 10/16/21
// Version              : LED
// Target MCU           : AVR128DB48
// Target Hardware      ; none
// Author               : Vanessa Li
// DESCRIPTION
// Increments seconds counter and checks if seconds counter is equal
// to the clean variable. If it is equal, the "clean" LED will light
// up, else the "fill" LED will light up.
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
ISR(TCA0_OVF_vect) {
        second_counter += 0x01; // increment seconds counter

        if(TCA0_SINGLE_PER > 10000){
         printf("second_counter=%d\n", second_counter);
         printf("clean_time=%d\n", clean_time);
         printf("delay_end=%d\n", delay_end);
         printf("mode=%c\n", mode);
        }else if(second_counter%30 == 0){
        printf("second_counter=%d\n", second_counter);
         printf("clean_time=%d\n", clean_time);
         printf("delay_end=%d\n", delay_end);
         printf("mode=%c\n", mode);
        }else if(mode == 'd'){
         printf("mode=%c\n", mode);
        }

        TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm; //clear interrupt flags
}

//***************************************************************************
//
// Function Name        : "port_init"
// Date                 : 9/30/21
// Version              : 1.2
// Target MCU           : AVR128DB48
// Target Hardware      ; Push Button
// Author               : Brandon Guzy
// DESCRIPTION
// This program sets up the ports of the AVR128DB28 as inputs, enables
// pull ups, and interrupts. At the end of the subroutine, interrupts are
// enabled
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : v1.1 Updated code to clear accidental interrupts
//                        after initialization
//                        v1.2 Updated to interrupt on falling edge to prevent
//                        many interrupts when holding button
//
//**************************************************************************
void port_init(void){
        PORTC.DIR &= 0xF0; //C0-3, inputs for LCD pushbuttons
        PORTA.DIR |= 0b00001100; //A2-3, outputs for SSR enables(fill,clean)
        PORTD.DIR |= 0b10000000; //D7, output for SSR enable(BB2)
        PORTF.DIR &= 0b11111100; //F0-1, inputs for external pushbuttons

        PORTF.PIN0CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        PORTF.PIN1CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        PORTC.PIN0CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        PORTC.PIN1CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        PORTC.PIN2CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        PORTC.PIN3CTRL = 0b00001011; //enable pull up and interrupt on falling edge
        _delay_ms(50);               //wait for pull ups and interrupts to fully enable

        PORTC_INTFLAGS = 0xFF;      //clear accidental interrupt flags
```

```
}

//during a fill, fill valve and BB2 valve are open
void start_fill(void) {
	mode = 'f';
	PORTA.OUT |= 0b00000100; //open fill valve
	PORTD.OUT |= 0b10000000; //open BB2 valve
	if(clean == 1) { //fill duration is 45 sec if after clean event
		delay_end = second_counter + fill_delay;
	}else {		//fill duration defaults to 20 secs if top off fill
		delay_end = second_counter + topOff_fill_delay;
	}

}

//during clean, clean valve and BB2 valve are open, then BB2 closes after 15 secs
void start_clean(void) {
	clean = 1; //set clean to 1 so that the system knows that a clean event occurred
	mode = 'c';
	PORTA.OUT |= 0b00001000; //open clean valve
	PORTD.OUT |= 0b10000000; //open BB2 valve
	delay_end = second_counter + clean_delay;
}

//cancel_fill and cancel_clean are used to cancel clean/fill during the event by pressing
any of the pushbuttons for the LCD
void cancel_fill(void) {
	PORTA.OUT &= 0b11111011;
	PORTD.OUT &= 0b01111111;
	//go back to disabled menu if previously disabled
	mode = ((disabled == 1) ? 'd' : 'h');
	if(clean == 1) { //if clean event was before filling, reset second_counter to 0
		second_counter = 0;
		clean = 0;
	}
}

void cancel_clean(void) {
	PORTA.OUT &= 0b11110111;
	PORTD.OUT &= 0b01111111;
	second_counter = 3600;		//skip to fill event
}

int main(void) {
	init_lcd_dog();
	snprintf(dsp_buff1, 21, "Startup Process");
	snprintf(dsp_buff2, 21, "-----------------");
	snprintf(dsp_buff3, 21, "------------------");
	snprintf(dsp_buff4, 21, "-------------------");
	update_lcd_dog();
	port_init();
	TCA0_init();
	USART0_init();
	ADC0_init();
	sei();
	while(1) {
		if(second_counter%3600 == 0 && second_counter != clean_time &&
		second_counter >= 3600){
        start_fill();
}else if(second_counter == clean_time){
        start_clean();
}

if(solarConversion() < 12.0 && night_mode == 1) {
        mode = 'g';
}

switch(mode){
        case 'd': //disabled menu
                snprintf(dsp_buff1, 21, "                    ");
                snprintf(dsp_buff2, 21, "   SYSTEM DISABLED  ");
                snprintf(dsp_buff3, 21, "                    ");
                snprintf(dsp_buff4, 21, "               ENBL ");
                second_counter = 0;
                disabled = 1;
        break;
        case 'h'://home menu
                PORTA.OUT &= 0b11111011;
                PORTD.OUT &= 0b01111111;
                PORTA.OUT &= 0b11110111;
                disabled = 0;
                snprintf(dsp_buff1, 21, "LAST NEXT NEXT    NM");
                if(second_counter < 3600){
                        snprintf(dsp_buff2, 21, "CLN  FILL FILL   %s ",
                        (night_mode == 1) ? " ON" : "OFF");
                        snprintf(dsp_buff3, 21, "%d:%02d 0:%02d 1:%02d      ",
                        (second_counter)/3600, (second_counter%3600)/60,
                        (3600 - (second_counter%3600))/60,
                        (3600 - (second_counter%3600))/60);
                }else if(abs(clean_time - second_counter) < 3600) {
                        snprintf(dsp_buff2, 21, "FILL CLN  FILL   %s ",
                        (night_mode == 1) ? " ON" : "OFF");
                        snprintf(dsp_buff3, 21, "%d:%02d 0:%02d 1:%02d      ",
                        (second_counter%3601)/3600, (second_counter%3600)/60,
                        (3600 - (second_counter%3600))/60,
                        (3600 - (second_counter%3600))/60);
                }else if(clean_time - second_counter < (3600*2) && clean_time
                - second_counter > 3600) {
                        snprintf(dsp_buff2, 21, "FILL FILL CLN    %s ",
                        (night_mode == 1) ? " ON" : "OFF");
                        snprintf(dsp_buff3, 21, "%d:%02d 0:%02d 1:%02d      ",
                        (second_counter%3601)/3600, (second_counter%3600)/60,
                        (3600 - (second_counter%3600))/60,
                        (3600 - (second_counter%3600))/60);
                }else if(clean_time - second_counter > (3600*2)) {
                        snprintf(dsp_buff2, 21, "FILL FILL FILL   %s ",
                        (night_mode == 1) ? " ON" : "OFF");
                        snprintf(dsp_buff3, 21, "%d:%02d 0:%02d 1:%02d      ",
                        (second_counter%3601)/3600, (second_counter%3600)/60,
                        (3600 - (second_counter%3600))/60,
                        (3600 - (second_counter%3600))/60);
                }
                snprintf(dsp_buff4, 21, "MODE DIAG CLEAN FILL");
                break;
        case 'l'://schedule clean menu
                snprintf(dsp_buff1, 21, "Fill Every Hour     ");
        snprintf(dsp_buff2, 21, "How many Fills?      ");
        snprintf(dsp_buff3, 21, "%d Fills per 1 Clean ",
        clean_time/3600 - 1);
        snprintf(dsp_buff4, 21, "INC  DEC  CLEAN HOME ");
        break;
case 'i'://schedule fill menu
        snprintf(dsp_buff1, 21, "Duration of fill?    ");
        snprintf(dsp_buff2, 21, "%dm %ds              ",
        topOff_fill_delay/60, topOff_fill_delay%60);
        snprintf(dsp_buff3, 21, "                     ");
        snprintf(dsp_buff4, 21, "INC  DEC  HOME  FILL ");
        break;
case 'e'://disable system menu
        snprintf(dsp_buff1, 21, "DISABLE SYSTEM?     ");
        snprintf(dsp_buff2, 21, "                    ");
        snprintf(dsp_buff3, 21, "                    ");
        snprintf(dsp_buff4, 21, "YES    NO           ");
        break;
case 'n'://enable night mode menu
        if(night_mode == 1) {
                snprintf(dsp_buff1, 21, "Turn off night mode? ");
        }else if(night_mode == 0){
                snprintf(dsp_buff1, 21, "Turn on night mode?  ");
        }
        snprintf(dsp_buff2, 21, "                     ");
        snprintf(dsp_buff3, 21, "                     ");
        snprintf(dsp_buff4, 21, "YES    NO           ");
        break;
case 'f'://fill menu
        snprintf(dsp_buff1, 21, "Currently Filling    ");
        snprintf(dsp_buff2, 21, "Time Remaining: %d:%02d ",
        (delay_end - second_counter)/ 60, (delay_end - second_counter)
        %60);
        snprintf(dsp_buff3, 21, "                     ");
        snprintf(dsp_buff4, 21, "Any Button to Cancel ");
        if(second_counter >= delay_end) { //end fill
                PORTA.OUT &= 0b11111011;
                PORTD.OUT &= 0b01111111;
                //go back to disabled menu if previously disabled
                mode = ((disabled == 1) ? 'd' : 'h');
        //if clean event was before filling, reset second_counter to 0
                if(clean == 1) {
                        second_counter = 0;
                        clean = 0;
                }
        }
        break;
case 'c'://clean menu
        snprintf(dsp_buff1, 21, "Currently Cleaning   ");
        snprintf(dsp_buff2, 21, "Time Remaining: %d:%02d ", (delay_end
        - second_counter)/ 60, (delay_end - second_counter) %60);
        snprintf(dsp_buff3, 21, "                      ");
        snprintf(dsp_buff4, 21, "Any Button to Cancel  ");
        //close BB2 after 15 seconds
        if (delay_end - second_counter <= 30){
                PORTD.OUT &= 0b01111111;
        }
        if(second_counter >= delay_end) { //turn off clean valve
```

```
                    PORTA.OUT &= 0b11110111;
                    second_counter = 3600;      //skip to fill event
                    start_fill();
                }
                break;
            case 'm'://mode menu
                snprintf(dsp_buff1, 21, "Select an option:   ");
                snprintf(dsp_buff2, 21, "                    ");
                //Print IP address
                snprintf(dsp_buff3, 21, "%s                    ", IPAdd);
                snprintf(dsp_buff4, 21, "DSBL  NM        HOME");
                break;
            case 'a'://diagnostics menu
                {
                    runDiagnostics();
                    int tempInt1 = AIN1;
                    double tempFrac1 = AIN1 - tempInt1;
                    int fracInt1 = trunc(tempFrac1 * 100);
                    int tempInt2 = AIN2;
                    double tempFrac2 = AIN2 - tempInt2;
                    int fracInt2 = trunc(tempFrac2 * 100);
                    int tempInt3 = AIN3;
                    double tempFrac3 = AIN3 - tempInt3;
                    int fracInt3 = trunc(tempFrac3 * 100);
                    snprintf(dsp_buff1, 21, "SSR1 SSR2  SSR3  SOL");
                    snprintf(dsp_buff2, 21, "%d.%02d %d.%02d  %d.%02d",
                    tempInt1, fracInt1, tempInt2, fracInt2, tempInt3,
                    fracInt3);
                    snprintf(dsp_buff3, 21, "                    ");
                    snprintf(dsp_buff4, 21, "                HOME");
                }
                break;
            case 'g': //night mode menu
                snprintf(dsp_buff1, 21, "                     ");
                snprintf(dsp_buff2, 21, " NIGHT MODE ENABLED  ");
                snprintf(dsp_buff3, 21, "                     ");
                snprintf(dsp_buff4, 21, "                DSBL ");
                second_counter = 0;
                //disable interrupt for external pushbuttons
                PORTF.PIN0CTRL = 0b00001100;
                PORTF.PIN1CTRL = 0b00001100;
                if(solarConversion() >= 12) {
                    mode = 'h';
                }
                break;
        }
        update_lcd_dog();
    }

}
```

USART_config

```
#ifndef USART_CONFIG_H_
#define USART_CONFIG_H_

//set baud rate
#define USART0_BAUD_RATE(BAUD_RATE) ((float)(F_CPU * 64 / (16 *(float)BAUD_RATE)) + 0.5)
void USART0_init(void);
int USART0_printChar(char character, FILE *stream);

#endif /* USART_CONFIG_H_ */

/*define the USART0_printchar function early
so it can be used in the USART_stream declaration*/
int USART0_printChar(char character, FILE *stream);

//setup a stream to print to USART, This will be used in place of stdout
FILE USART_stream = FDEV_SETUP_STREAM(USART0_printChar, NULL, _FDEV_SETUP_WRITE);

//variables for USART0 Reading
char command[50];
uint8_t cmd_index = 0;
char c;

//***************************************************************************
//
// Function Name        : "USART0_init"
// Date                 : 11/12/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; USART0 general output
// Author               : Brandon Guzy
// DESCRIPTION
// This sets up the USART on pins A0 and A1 so they are ready to transmit
// and receive data
// also replaces stdout with the USART stream so printf can be used to output
// to USART
//
// Warnings             : none
// Restrictions         : requires that USART0_BAUD_RATE function be defined
//                        requires that USART_stream be defined
// Algorithms           : none
// References           : none
//
// Revision History     : v1.0: Initial version
//                        v1.1: Updated to enable receive complete interrupt
//                              and enable receive functionality
//
//**************************************************************************
void USART0_init(void)
{
	PORTA.DIR &= ~PIN1_bm;
	PORTA.DIR |= PIN0_bm;

	USART0.BAUD = (uint16_t)USART0_BAUD_RATE(115200);
	USART0.CTRLA |= USART_RXCIE_bm;
	USART0.CTRLB |= USART_TXEN_bm;
	USART0.CTRLB |= USART_RXEN_bm;     //enable receive
	stdout = &USART_stream;
}

//***************************************************************************
//
// Function Name        : "USART0_RXC_vect Interrupt"
// Date                 : 12/5/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; USART0 input
// Author               : Brandon Guzy
// DESCRIPTION
// This runs every time data is received over USART. For this project, it is
// character data. The character data is stored in a temporary string.
// When the newline character is received, the string is presumed to be
// complete, and the execute_USART_command function is run
//
// Warnings             : none
// Restrictions         : Requires global variables char[] command,
//                          int cmd_index, char c to be defined
//
// Algorithms           : none
// References           : execute_USART_command()
//
// Revision History     : Initial version
//
//**************************************************************************
ISR(USART0_RXC_vect){
	while (!(USART0.STATUS & USART_RXCIF_bm))//wait for data to be available
	{
		;
	}
	c = USART0.RXDATAL;
	if(c != '\n' && c != '\r')
	{
		command[cmd_index++] = c;
		if(cmd_index > 50)
		{
			cmd_index = 0;
		}
	}
	if(c == '\n')
	{
		command[cmd_index] = '\0';
		cmd_index = 0;
		execute_USART_command(command);
	}
}


//***************************************************************************
//
// Function Name        : "USART0_printChar"
// Date                 : 11/12/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; USART general output
// Author               : Brandon Guzy
// DESCRIPTION
// This prints out a character from a file stream to USART0
// This will be used in conjunction with other code to print from stdout
// to USART
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
int USART0_printChar(char character, FILE *stream)
{
       while (!(USART0.STATUS & USART_DREIF_bm))
       {
              ;
       }
       USART0.TXDATAL = character;
       return 0;

}
```

DOG204_LCD

```
#ifndef DOG204_LCD_H_
#define DOG204_LCD_H_

//DOG LCD buffer
char dsp_buff1[21];
char dsp_buff2[21];
char dsp_buff3[21];
char dsp_buff4[21];

void lcd_spi_transmit_CMD (unsigned char cmd);
void lcd_spi_transmit_DATA (unsigned char cmd);
void init_spi_lcd (void);
void init_lcd_dog (void);
void delay_40mS(void);
void delay_30uS(void);
void update_lcd_dog(void);
void SPI0_wait();



#endif /* DOG204_LCD_H_ */

//***************************************************************************
//
// Function Name : "SPI_wait"
// Date : 9/14/21
// Version : 1.0
// Target MCU : AVR128DB48
// Target Hardware ;
// Author : Brandon Guzy
// DESCRIPTION
// This function waits until the interrupt flags for SPI are clear
//before proceeding
//
// Warnings : none
// Restrictions : none
// Algorithms : none
// References : none
//
// Revision History : Initial version
//
//**************************************************************************
void SPI0_wait(){
	//wait until there is no data waiting to be written or read
	while((SPI0.INTFLAGS & 0b11000000) != 0x80 ){
		asm volatile("nop");
	}
}

void delay_40mS(void){
	_delay_ms(40);
}

void delay_30uS(void){
	_delay_us(50);
}

//***************************************************************************
//
// Function Name        : "lcd_spi_transmit_DATA"
// Date                 : 9/14/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; ST7036 + LCD
// Author               : Brandon Guzy
// DESCRIPTION
// This function transmits data to the LCD
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void lcd_spi_transmit_DATA (unsigned char cmd) {
	SPI0.DATA = 0x5F;    //Send 5 synchronization bits, RS = 1, R/W = 0
	SPI0_wait();
	SPI0.DATA = cmd & 0x0F;     //transmit lower data bits
	SPI0_wait();
	SPI0.DATA = ((cmd>>4) & 0x0F);     //send higher data bits
	SPI0_wait();
}

//***************************************************************************
//
// Function Name        : "lcd_spi_transmit_CMD"
// Date                 : 9/14/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; ST7036 + LCD
// Author               : Brandon Guzy
// DESCRIPTION
// This function transmits a command to the LCD, RS =0
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void lcd_spi_transmit_CMD (unsigned char cmd) {
       SPI0.DATA = 0x1F;    //Send 5 synchronisation bits, RS = 0, R/W = 0
       SPI0_wait();
       SPI0.DATA = cmd & 0x0F;     //transmit lower data bits
       SPI0_wait();
       SPI0.DATA = ((cmd>>4) & 0x0F);     //send higher data bits
       SPI0_wait();
}

//***************************************************************************
//
// Function Name        : "init_spi_lcd"
// Date                 : 9/14/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; ST7036 + LCD
// Author               : Brandon Guzy
// DESCRIPTION
// This program initializes all ports and peripherals
// necessary to communicate with the LCD.
// Does not initialize any values on the LCD, only the AVR128
// Uses the standard SPI pins (PA7,6,4) for SPI communication
// PA7 = Slave Select, PA4 = MOSI, PA6 = SCK
// and PC0 for command select, RS = 1 for data, RS = 0 for command
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void init_spi_lcd(){
       PORTA.DIR |= 0b11010000;    //enable output on necessary port A pins
       SPI0.CTRLA = 0b01100001;    //enable SPI, set in master mode, LSB First
       SPI0.CTRLB = 0b00000000;    //disable buffer, slave select enabled, normal mode
       PORTC.DIR = 0x01;                  //enable output on port C0
       PORTA.OUT &= 0b01111111;    //clear Pin A7 to reset.
       delay_30uS();
       PORTA.OUT |= 0x80;                 //set high to finish reset
}
```

```
//*****************************************************************************
//
// Function Name        : "init_lcd_dog"
// Date                 : 9/14/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; ST7036 + LCD
// Author               : Brandon Guzy
// DESCRIPTION
// This program initializes the DOG LCD so it is ready to be sent lines.
// This program sends several commands using the spi_transmit_CMD function
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//****************************************************************************
void init_lcd_dog (void) {
       init_spi_lcd();            //Initialize mcu for LCD SPI

       //start_dly_40ms:
       delay_40mS();    //startup delay.


       //func_set1:
       lcd_spi_transmit_CMD(0x3A);   //8-Bit data length extension Bit RE=1; REV=0


       //func_set2:
       lcd_spi_transmit_CMD(0x09); //4 line display



       //bias_set:
       lcd_spi_transmit_CMD(0x06); //Bottom view
       //delay_30uS();      //delay for command to be processed


       //power_ctrl:
       lcd_spi_transmit_CMD(0x1E); //Bias setting BS1=1
       //~ 0x55 for 3.3V (delicate adjustment).


       //follower_ctrl:
       lcd_spi_transmit_CMD(0x39); //8-Bit data length extension Bit RE=0; IS=1

       //contrast_set:
       lcd_spi_transmit_CMD(0x1B); //BS0=1 -> Bias=1/6


       //display_on:
       lcd_spi_transmit_CMD(0x6E); //Devider on and set value
```

```
	//clr_display:
	lcd_spi_transmit_CMD(0x57); //Booster on and set contrast (BB1=C5, DB0=C4)


	//entry_mode:
	lcd_spi_transmit_CMD(0x72); //Set contrast (DB3-DB0=C3-C0)

	//entry_mode:
	lcd_spi_transmit_CMD(0x38); //8-Bit data length extension Bit RE=0; IS=0

	//display_on
	lcd_spi_transmit_CMD(0x0C); //Display on, Cursor off, Blink off

	lcd_spi_transmit_CMD(0x01);		//clear display
}

//***************************************************************************
//
// Function Name        : "update_lcd_dog"
// Date                 : 9/14/21
// Version              : 1.0
// Target MCU           : AVR128DB48
// Target Hardware      ; ST7036 + LCD
// Author               : Brandon Guzy
// DESCRIPTION
// This program updates the LCD with the characters from the 3 line buffers
//
// Warnings             : none
// Restrictions         : none
// Algorithms           : none
// References           : none
//
// Revision History     : Initial version
//
//**************************************************************************
void update_lcd_dog(void) {

	// send line 1 to the LCD module.
	lcd_spi_transmit_CMD(0x80); //init DDRAM addr-ctr
	delay_30uS();
	for (int i = 0; i < 20; i++) {
		if(dsp_buff1[i] == '\0')
			break;
		lcd_spi_transmit_DATA(dsp_buff1[i]);
	}

	// send line 2 to the LCD module.
	lcd_spi_transmit_CMD(0xA0); //init DDRAM addr-ctr
	delay_30uS();
	for (int i = 0; i < 20; i++) {
		if(dsp_buff2[i] == '\0')
			break;
		lcd_spi_transmit_DATA(dsp_buff2[i]);
	}

	// send line 3 to the LCD module.
	lcd_spi_transmit_CMD(0xC0); //init DDRAM addr-ctr
	delay_30uS();
```

```
        for (int i = 0; i < 20; i++) {
                if(dsp_buff3[i] == '\0')
                        break;
                lcd_spi_transmit_DATA(dsp_buff3[i]);
        }
        lcd_spi_transmit_CMD(0xE0); //init DDRAM addr-ctr
        for (int i = 0; i < 20; i++) {
                if(dsp_buff4[i] == '\0')
                        break;
                lcd_spi_transmit_DATA(dsp_buff4[i]);
        }
}

void clear_display(void){
        lcd_spi_transmit_CMD(0x01);        //clear display

}
```

ADC_diagnostic

```
#ifndef ADC_DIAGNOSTIC_H_
#define ADC_DIAGNOSTIC_H_

double AIN1, AIN2, AIN3;

double ADCpinSel_and_output (uint8_t pinNum);
void ADC0_init(void);
void runDiagnostics(void);
double solarConversion(void);
void POST(void);

#endif /* ADC_DIAGNOSTIC_H_ */

double ADCpinSel_and_output (uint8_t pinNum){
        double adcVal;
        ADC0.MUXPOS = pinNum;
        ADC0.COMMAND = 0x01; // start the conversion
        while (!(ADC0.INTFLAGS & ADC_RESRDY_bm))
        {
                ;
        }
        adcVal = ADC0.RES/1600.0; // read the value into adcVal
        ADC0.INTFLAGS = ADC_RESRDY_bm; // clear the flag
        return adcVal;
}


void ADC0_init(void){
        VREF.ADC0REF = VREF_REFSEL_2V048_gc;
        ADC0.CTRLC = ADC_PRESC_DIV128_gc;
        ADC0.CTRLA |= ADC_ENABLE_bm;

        /* Disable interrupt and digital input buffer on PD4 */
        PORTD.PIN4CTRL &= ~PORT_ISC_gm;
        PORTD.PIN4CTRL |= PORT_ISC_INPUT_DISABLE_gc;
        /* Disable interrupt and digital input buffer on PD5 */
        PORTD.PIN5CTRL &= ~PORT_ISC_gm;
        PORTD.PIN5CTRL |= PORT_ISC_INPUT_DISABLE_gc;
        /* Disable interrupt and digital input buffer on PD6 */
        PORTD.PIN5CTRL &= ~PORT_ISC_gm;
        PORTD.PIN5CTRL |= PORT_ISC_INPUT_DISABLE_gc;
}

void runDiagnostics(void) {
        PORTA.OUT |= 0b00000100;
        PORTD.OUT |= 0b10000000;
        PORTA.OUT |= 0b00001000;
        AIN1 = ADCpinSel_and_output(0x04);
        AIN2 = ADCpinSel_and_output(0x05);
        AIN3 = ADCpinSel_and_output(0x06);
}

double solarConversion(void) {
        return ADCpinSel_and_output(0x03);
}

void POST(void) { //run power-on self-test
        runDiagnostics();
        snprintf(dsp_buff1, 21, "Power-On Self Tests ");
        snprintf(dsp_buff2, 21, "FILL-%s  WIFI-GOOD", ((AIN1 > 1) ? "GOOD" : "FAIL"));
        snprintf(dsp_buff3, 21, " CLN-%s  SOLR-%s", ((AIN3 > 1) ? "GOOD" : "FAIL"),

        ((solarConversion() > 1) ? "GOOD" : "FAIL"));
        snprintf(dsp_buff4, 21, " BB2-%s           ",((AIN2 > 1) ? "GOOD" : "FAIL"));
        update_lcd_dog();
        _delay_ms(10000);
```

Wi-Fi Code

Main Wi-Fi Code

```
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <ESPAsync_WiFiManager.h>           //https://github.com/khoih-prog/ESPAsync_WiFiManager

#include "homepage.h"

#define HTTP_PORT          80

const char* ssid     = "ESP8266-BBAC";  //Access Point SSID
const char* password = "";         //Access Point Password (none)

int prevMillis;

String serialStr = "";             //String Being Built from Serial

// Create AsyncWebServer object on port 80
AsyncWebServer server(HTTP_PORT);
DNSServer dnsServer;
AsyncWebSocket ws("/ws");
```

```
void handleRoot();              // function prototypes for HTTP handlers
void handleNotFound();

IPAddress local_IP(192,168,4,22);   //IP Address for the website
IPAddress gateway(192,168,4,9);     //config for the AP
IPAddress subnet(255,255,255,0);

//handle incoming data function not used yet
void handlingIncomingData(void *arg, uint8_t *data, size_t len) {
  AwsFrameInfo *info = (AwsFrameInfo*)arg;
  if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
    data[len] = 0;
    Serial.printf((char*)data);
  }
}

// Handler for incoming event
void onEvent(AsyncWebSocket * server, AsyncWebSocketClient * client, AwsEventType type,
             void * arg, uint8_t *data, size_t len){
             //mostly Debug messages
   switch(type) {
      case WS_EVT_CONNECT:
        Serial.printf("Client connected: \n\tClient id:%u\n\tClient IP:%s\n",
             client->id(), client->remoteIP().toString().c_str());
        break;
      case WS_EVT_DISCONNECT:
         Serial.printf("Client disconnected:\n\tClient id:%u\n", client->id());
         break;
      case WS_EVT_DATA:
                  //handle incoming data
         handlingIncomingData(arg, data, len);
         break;
      case WS_EVT_PONG:
          Serial.printf("Pong:\n\tClient id:%u\n", client->id());
          break;
      case WS_EVT_ERROR:
          Serial.printf("Error:\n\tClient id:%u\n", client->id());
          break;
   }

}

void setup() {
  Serial.begin(115200);   //Begin Serial Communications at 115200 baud
  delay(200);             //Short delay before transmitting
  Serial.println();       //Print a newline
  WiFi.setAutoReconnect(true);
  WiFi.persistent(true);
  WiFi.begin();
  if(WiFi.waitForConnectResult() != WL_CONNECTED){//Open WiFi Manager only if not connected
    Serial.println("Entered WiFi Config Mode");
    ESPAsync_WiFiManager ESPAsync_wifiManager(&server, &dnsServer);
    ESPAsync_wifiManager.setAPStaticIPConfig(local_IP, gateway, subnet);
    if(!ESPAsync_wifiManager.autoConnect("BBAC-CONFIG")){
      WiFi.softAPConfig(local_IP, gateway, subnet);   //set AP config with values from above
      WiFi.softAP("BBAC-HOSTED");                      //set AP SSID, with no password
    }
  }


  //on webserver event call the event handler
  ws.onEvent(onEvent);
  server.addHandler(&ws);

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){  //serve the homepage on HTTP GET
request
      request->send_P(200, "text/html", homepage, NULL);
  });

  server.begin();                          // Actually start the server

  prevMillis = 0;     //used for interrupt for sending IP address

  Serial.println("HTTP, mDNS server started");    //debug
  serialStr = "";
}

void loop(void){
  ws.cleanupClients();
  while (Serial.available() > 0) {
    // get the new byte:
    char inChar = (char)Serial.read();
    // check if it is the end of the string
    if(inChar == '\n'){
    //send the string over websocket
      ws.textAll(serialStr);
      //clear the string
    serialStr = "";
    }else{
    //else if not end of string just add the next character
      serialStr += inChar;
    }
  }
  if (millis() - prevMillis >= 5000){   //if more than 5s has elapsed since last print, send the IP address again
    Serial.print("IP:");
    Serial.println(WiFi.localIP());
    prevMillis = millis();
  }
}
```

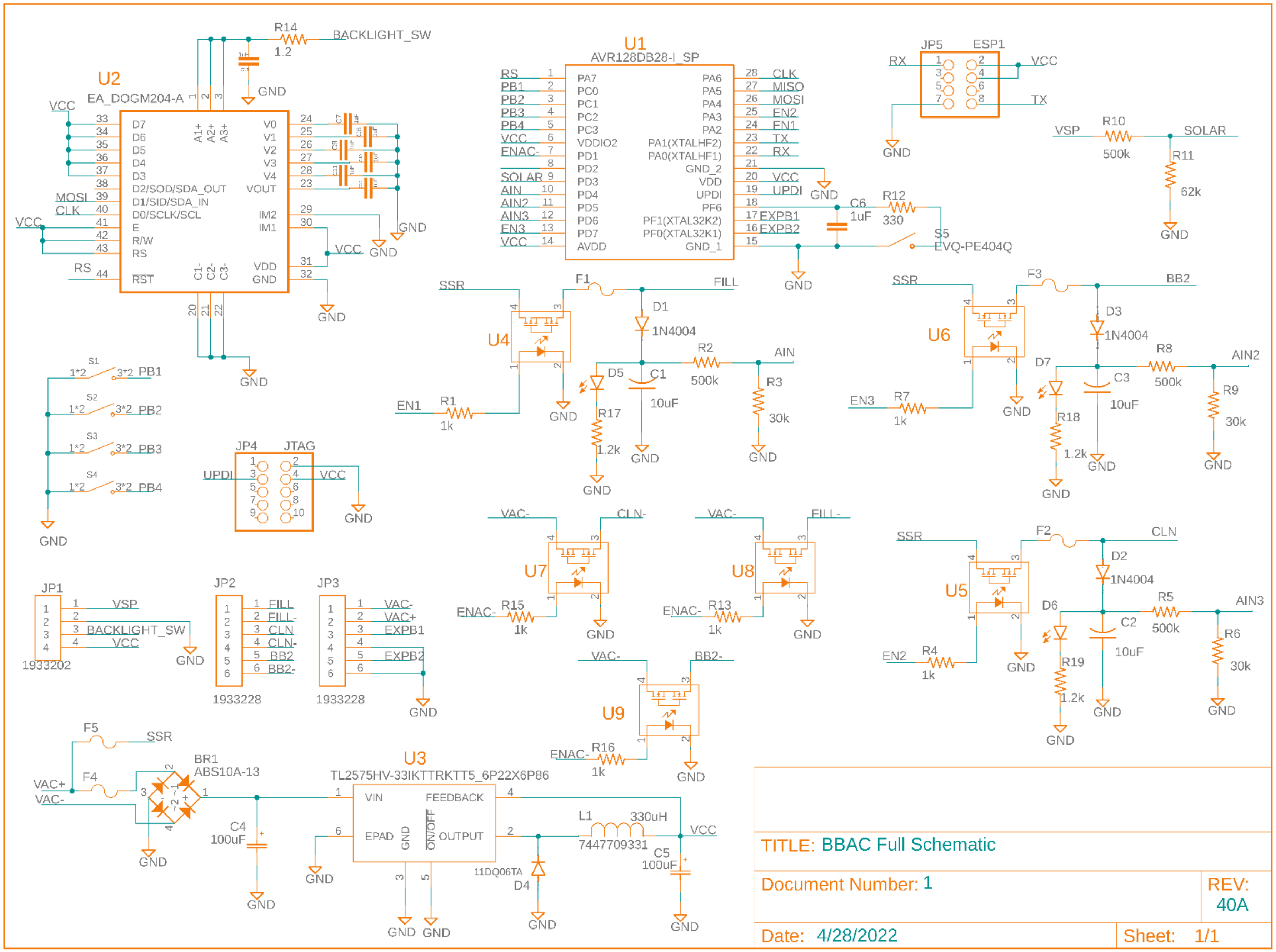
Webpage Code

```
const char homepage[] PROGMEM = R"=====(
<!DOCTYPE html>
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<html>
  <head>
    <style>
    table {
      border-collapse: collapse;
      width: 100%;
    }

    th, td {
      padding: 8px;
      text-align: center;
      border-bottom: 1px solid #ddd;
    }
    button{
      width: 75%;
      max-width: 250px;
      height:50px;
      font-size:14pt;
    }
    </style>
    </head>
<body style="background-color: #0021df; color:#ddd ">
<center>
<h1>BBAC Remote Server</h1>
<h3 id ="status-indicator" style = "color:rgb(255, 95, 95)">Status: Disconnected</h4>

 <div id = 'next-show'>
  <h3 id="next-fill">Next Fill in:</h3>
  <h3 id= "next-clean">Next Clean in:</h3>
 </div>

 <div id = 'diagnostic-show' style = "display:none">
  <h2 style="color: rgb(255, 255, 255);">Diagnostics Running</h2>
  <button type = "button" id = "cancel-diag">Cancel Diagnostics</button>
  <br>
</div>

 <div id = 'clean-fill' style = "display:none">
  <h2 id = 'curr-mode' style="color: rgb(255, 255, 255);">CURRENTLY FILLING</h2>
  <h2 id = 'clean-fill-timer' style="color: rgb(255, 255, 255);">Timer Here</h2>
  <button type = "button" id = "cancel-button">Cancel Operation</button>
 </div>

<div id = "disable-show">
  <h2>System In Disable Mode</h2>
  <button type = "button" id = "cancel-disable" >Enable System</button>
</div>

 <div id = "clean-fill-buttons" style = "display:block">
  <button type = "button" id = "fill-button" >Start Fill Event</button>
  <br>
  <br>
  <button type = "button" id = "clean-button" >Start Clean Event</button>
  <br>
  <br>
  <button type = "button" id = "disable-button" >Disable System</button>
 </div>

<table id="serialData" style = "display:none">
  <tr><th>Serial Data Goes Below</th></tr>
```

```
</table>
</center>
<script language="javascript">
 //do work
 //start websocket with the same URL as this webpage
 var gwUrl = "ws://" + location.host + "/ws";
   var webSocket = new WebSocket(gwUrl);
   document.getElementById('clean-button').addEventListener("click", cleanClick);
   document.getElementById('fill-button').addEventListener("click", fillClick);
   document.getElementById('cancel-button').addEventListener("click", cancelClick);
   document.getElementById('cancel-disable').addEventListener("click", cancelClick);
   document.getElementById('disable-button').addEventListener("click", disableClick);
 //create an object with all variables
 //this was created for dynamic access
 //this means that the variable name can be specified by a string
   var valObj = {
     second_counter: 0,
     mode: 'h',
     clean_time: 0,
     delay_end: 0
   }

   webSocket.onopen = function(event) {
       console.log("open"); //log to JS console
       if(webSocket.readyState === WebSocket.OPEN){
         document.getElementById('status-indicator').style.color= 'rgb(113, 255, 74)';
         document.getElementById('status-indicator').innerHTML = "Status: Connected";
       }


   }
   webSocket.onclose = function(event) {
     console.log('Socket is closed. Reconnect will be attempted in 1 second.', e.reason);


   }

   function checkWSStatus(){
     if(webSocket.readyState === WebSocket.OPEN){
       document.getElementById('status-indicator').style.color= 'rgb(113, 255, 74)';
       document.getElementById('status-indicator').innerHTML = "Status: Connected";
     }else{
       document.getElementById('status-indicator').style.color= 'rgb(255, 95, 95)';
       document.getElementById('status-indicator').innerHTML = "Status: Disconnected";
     }
   }

   function cleanClick(){
     console.log("clean button clicked");
     webSocket.send("clean\n");
   }

   function fillClick(){
     console.log("fill button clicked");
     webSocket.send("fill\n");
   }
```

```
    function cancelClick(){
      console.log("cancel button clicked");
      webSocket.send("cancel\n");
    }

    function disableClick(){
      console.log("disable button clicked");
      webSocket.send("disable\n");
    }

    function hideAll(){
      document.getElementById("clean-fill").style.display = "none";
      document.getElementById("clean-fill-buttons").style.display = "none";
      document.getElementById("next-show").style.display = "none";
      document.getElementById("disable-show").style.display = "none";
    }

    webSocket.onmessage = function(event) {
      //console.log("message"); //debug message
      var data = event.data;
      var table = document.getElementById("serialData");
      splitDat = data.split('='); //split the message by = sign
      valObj[splitDat[0]] = splitDat[1];
      /*if(Boolean(data)){  //insert the message into the table for debugging
        var row = table.insertRow(1);
        var cell = row.insertCell(0);
        cell.innerHTML = data;
      }*/
    //update next clean timer
      document.getElementById("next-clean").innerHTML = "Clean In: " +
        Math.trunc((valObj['clean_time'] - valObj['second_counter'] )/3600) +
        ":" + Math.trunc(((valObj['clean_time'] -valObj['second_counter']
  )%3600)/60).toString().padStart(2,'0') + ":"
        + Math.trunc((valObj['clean_time'] -valObj['second_counter'] )%60).toString().padStart(2,'0');
    //update next fill timer
      document.getElementById("next-fill").innerHTML = "Fill In: 0:" +
        Math.trunc((3600 - (valObj['second_counter']  % 3600))/60).toString().padStart(2,'0')
        + ":" + Math.trunc((valObj['clean_time'] -valObj['second_counter'] ) %60).toString().padStart(2,'0');
    //check if in fill or clean mode
      if(valObj['mode'] == 'f' || valObj['mode'] == 'c'){
    //show the filling/cleaning div
        hideAll();
        document.getElementById("clean-fill").style.display = "block";

    //replace the text inside the clean/fill mode
        if(valObj['mode'] == 'f'){
          document.getElementById("curr-mode").innerHTML = "CURRENTLY FILLING";
        }else{
          document.getElementById("curr-mode").innerHTML = "CURRENTLY CLEANING";
        }
        //update clean/fill timer
        document.getElementById("clean-fill-timer").innerHTML = "Time Remaining: " +
          Math.trunc((valObj['delay_end']-valObj['second_counter'])/60) +
          ":" + Math.trunc((valObj['delay_end'] - valObj['second_counter']) % 60);
    //flash red and white on alternating seconds
        if(valObj['second_counter']%2 == 0){
          document.getElementById("curr-mode").style.color = 'rgb(255, 0, 0)';
        }else{
          document.getElementById("curr-mode").style.color = 'rgb(255, 255, 255)';
        }

      //if in disabled mode show nothing but the disable options
      }else if(valObj['mode'] == 'd'){
        hideAll();
        document.getElementById("disable-show").style.display = "block";
      }else{
    //hide the fill/clean div when not in fill/clean mode
        hideAll();
        document.getElementById("clean-fill-buttons").style.display = "block";
        document.getElementById("next-show").style.display = "block";
      }
    }


  </script>
  </body>
  </html>
  )=====
```

Appendix F: Full Schematic



U1
AVR128DB28-I_SP
U2
EA_DOGM204-A
U3
TL2575HV-33IKTTRKTT5_6P22X6P86
U4
U5
U6
U7
U8
U9
R14 1.2
BACKLIGHT_SW
JP5 ESP1
RX
TX
VCC
VSP
R10 500k
SOLAR
R11 62k
C6 1uF
R12 330
S5 EVQ-PE404Q
SSR
F1
FILL
D1 1N4004
R2 500k
AIN
R3 30k
C1 10uF
D5
R17 1.2k
EN1
R1 1k
F3
BB2
D3 1N4004
R8 500k
AIN2
R9 30k
C3 10uF
D7
R18 1.2k
EN3
R7 1k
F2
CLN
D2 1N4004
R5 500k
AIN3
R6 30k
C2 10uF
D6
R19 1.2k
EN2
R4 1k
VAC-
CLN-
FILL-
BB2-
ENAC-
R15 1k
R13 1k
R16 1k
S1
S2
S3
S4
PB1
PB2
PB3
PB4
JP4 JTAG
UPDI
JP1
1933202
JP2
1933228
JP3
1933228
CLN
EXPB1
EXPB2
VAC+
F4
F5
BR1 ABS10A-13
C4 100uF
L1 330uH
7447709331
C5 100uF
D4 11DQ06TA
VIN
FEEDBACK
EPAD
GND
ON/OFF
OUTPUT
GND
TITLE: BBAC Full Schematic
Document Number: 1
REV: 40A
Date: 4/28/2022
Sheet: 1/1