# Weather Adaptive Windows
# (Originally FA17 #27 and FA19#22)

By
Canlin Zhang
Kaiwen Zhao
Yihao Deng

Final Report for ECE445, Senior Design, Spring 2020
TA: Yichi Zhang

May 8, 2020
Project No.53

## Abstract

This document primarily describes a different approach to two previous projects regarding window automation in details. To deal with different weather conditions and make automated adjustments on these windows, both projects from fall 2017 #27 and fall 2019 #22 emphasized on sensor-based designs, whereas our approach focuses on online weather data gathering and processing, with an addition of a self-controlled shade to the mechanical system.

# Contents


1. Second Project Motivation 3

1.1. Updated Problem Statement 3

1.2. Updated Solution 3

1.3. Updated High-Level Requirements 4

1.4. Updated Visual Aid 4

1.5. Updated Block Diagram 4

2. Second Project Implementation 5

2.1. Implementation Details and Analysis 5

3. Second Project Conclusions 18

3.1 Implementation Summary 18

3.2 Unknows, uncertainties, testing needed 18

3.3 Ethics and Safety 18

3.4 Project Improvements 19

4. References 20

# 1. Second Project Motivation

## 1.1 Updated Problem Statement

Opening and closing windows may seem to be a trivial job in our daily life. However, if people forget to adjust the position of windows in some events like storms, it could lead to serious damage to their properties by flooding. In this case, if the windows can function automatically, it would mitigate the risk of flooding by closing up before storming. Moreover, it would be very convenient and comfortable for homeowners if the windows can adjust position themselves to regulate the room temperature by allowing a certain amount of breeze to enter the room. This would come in handy when people are sleeping at night. As being integral to the smart-home technology, windows also need to be developed in a way that it will work in a Wi-Fi network.

We are living in an era that every piece of furniture is starting to be redesigned into IOT devices with some extent of automation and ability to be connected with the IOT network. The idea to integrate windows and shades with some level of automation is clearly not an exotic idea given that many people now have the enthusiasm to make everything automated. There is also a report showing that automation of windows would have a market worth millions of dollars[1]. Nevertheless, there are only a few solutions for automated windows while all the components needed could be found on the market for reasonable cost. Therefore, we would like to design our own solution for an automated window and shade system.

## 1.2 Updated Solution

For the essential function such as closing the window for severe weather conditions and indoor temperature regulation, the original solutions utilized various sensors in the implementation to acquire real-time weather data for self-adjustment of the window. However, it would bring up the overall cost as the sensors are exposed in the outdoor area and will wear down or get lost during certain weather conditions like storming. So to design a low-cost and automated window with addition of easy accessibility to modern wireless network and smartphone technology, we propose a weather-adaptive window that is able to communicate wirelessly and automatically with application programming interfaces - a tool that can request and process online weather data. In this way, our windows can make adjustments based on the information gathered online without any sensors exposing in the outdoors.

In regarding the accuracy without using any temperature and rain sensors, it has been shown that a 5-day forecast has an accuracy of 90%. [2] In our case, we are basically using real-time weather data if the data are fetched constantly , which guarantees almost 100% accuracy. In contrast, even professional, agricultural use rain sensors have wide variances in terms of accuracies, from 27% to 97%. [3] Along with window adjustment, the automated white plastic shade would greatly reflect most of the radiation from the sun without trapping these radiation inside the room. This makes a huge difference in the room temperature. As mentioned in the problem statement, the automated windows are available on the market but not for a low cost and they are mostly only associated with remote controls. Homeowners, who wish to pay less and be relaxed when the change of weather condition occurs, might be interested in our solution.

## 1.3 Updated High-Level Requirements

- The control subsystem can correctly fetch local weather data from the Internet and process the data into according commands.
- The user can view the status of the window and control the angle of the window as well as the angle of shade slats from an Android application.
- The window can open/close to specific angles and the shade slats can adjust to angles according to local weather data from the Internet based on the user's location.

## 1.4 Updated Visual Aid

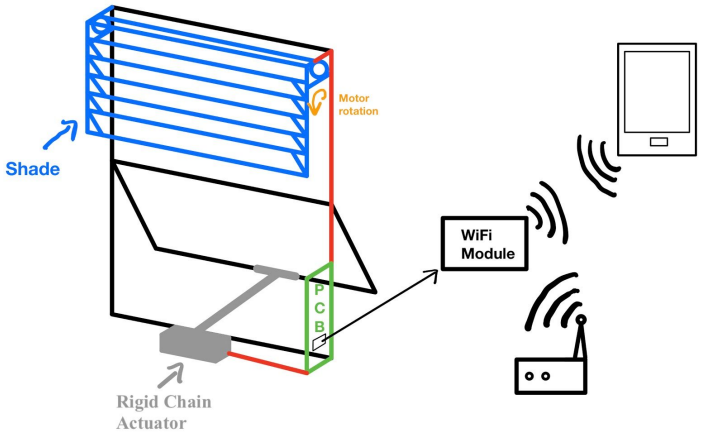



Figure: Updated visual aid

## 1.5 Updated Block Diagram

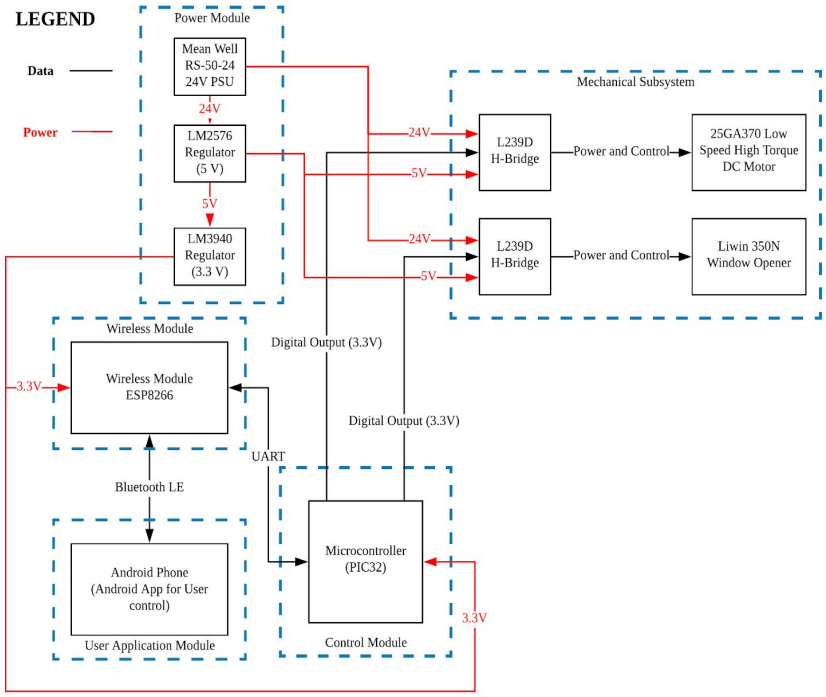



Figure: Updated block diagram

# 2. Second Project Implementation

## 2.1 Implementation Details and Analysis

### 2.1.1 MCU Software

The MCU software serves a series of functionalities based on three operation modes: Automatic mode, Manual mode and Configuration mode. In short, the MCU either accepts user configurations, automatically controls the window actuator and shades motor or accepts user controls over these mechanical components.

The details of the operation modes are listed in the following table:

| Operation Mode | Sequence of operations |
|---|---|
| Automatic Mode | 1. The MCU connects to the Internet through the user's home network using credentials provided by the user.<br>2. The MCU fetches local weather data according to current location.<br>3. The MCU parses fetched weather data and controls the operation of the window actuator and shades motor based on current weather data and user-defined parameters. |
| Manual Mode | 1. The MCU creates its own wireless network using the wireless controller.<br>2. The user connects his/her Android Phone to the wireless network created by MCU.<br>3. The MCU fetches user commands through the wireless network.<br>4. The MCU controls the operation of the window actuator and the shades motor based on user commands. |
| Configuration Mode | 1. The MCU creates its own wireless network using the wireless controller.<br>2. The user connects his/her Android Phone to the wireless network created by MCU.<br>3. The MCU fetches user configuration inputs through the wireless network.<br>4. The MCU writes user configuration data into non-volatile memory, such as a micro-SD card or onboard FLASH memory. |

The user would select from Auto mode and Configuration or Manual mode by flipping a switch on the MCU board.

We chose the way of mode change in this way since the major difference between Auto mode and other modes is the operation mode of the wireless controller: In Automatic mode, the MCU is connected to the user's home network, while in other modes, the MCU would generate its own wireless network using the wireless controller.

The user would select from Manual mode or Configuration mode in the Android application, after the user's Android phone is connected to the wireless network created by the wireless controller. This is because there is little difference between Manual mode and Configuration mode, since both modes basically accept user input, and the only work needs to be done to differentiate these two modes is to identify different data packets that resemble different modes.

Our current implementation prevents the user from switching between Auto mode and Configuration / Manual mode on the fly. That is, the user has to start from the Power On / Reset state of the microcontroller and flip the switch on the MCU board to change between modes. This is to prevent the user from changing the operation mode during the execution of the operations of other modes, which may cause potential errors.

Here is the list of implementations of some of the critical software operations in the MCU software.

**1. Wireless module initialization.**

The wireless module (ESP8266) connects to the MCU board using the UART interface. Therefore, the initialization of the wireless module mainly consists of UART initialization:

a. Initialization of the UART module on board, including selecting which UART interface to use. (Choose from UART 1/2/3, etc.)
b. Setting FIFO mode of UART. (Enabling interrupt-driven data receive/send or polling-driven data receive/send)
c. Setting line control mode of UART (Data size, parity bits, stop bits)
d. Set the BAUD rate of UART interface, the desired BAUD rate for ESP8266 wireless module is 115200. [4]
e. Enable specified UART interface. (Enable Rx/Tx/Peripheral)
f. Enabling interrupt from RX/TX pins. (If UART interrupts are enabled)

An example code of UART 2 initialization would look like this: [5]

```
  // This initialization assumes 36MHz Fpb clock. If it changes,
  // you will have to modify the baud rate initializer.
  UARTConfigure(UART2, UART_ENABLE_PINS_TX_RX_ONLY);
  UARTSetFifoMode(UART2, UART_INTERRUPT_ON_TX_NOT_FULL |
UART_INTERRUPT_ON_RX_NOT_EMPTY);
  UARTSetLineControl(UART2, UART_DATA_SIZE_8_BITS | UART_PARITY_NONE |
UART_STOP_BITS_1);
  UARTSetDataRate(UART2, GetPeripheralClock(), 115200);
  UARTEnable(UART2, UART_ENABLE_FLAGS(UART_PERIPHERAL | UART_RX | UART_TX));

  // Configure UART2 RX Interrupt
  INTEnable(INT_SOURCE_UART_RX(UART2), INT_ENABLED);
  INTSetVectorPriority(INT_VECTOR_UART(UART2), INT_PRIORITY_LEVEL_2);
  INTSetVectorSubPriority(INT_VECTOR_UART(UART2), INT_SUB_PRIORITY_LEVEL_0);
```

```
// configure for multi-vectored mode
INTConfigureSystem(INT_SYSTEM_CONFIG_MULT_VECTOR);
// enable interrupts
INTEnableInterrupts();
```

**2. Reading digital inputs.**

Reading digital inputs is relatively trivial. The program needs to:

a. Set the mode of the pins to digital I/O
b. Set the used pin(s) as digital input(s).
c. Read the logic value directly from pin value bits.

An example code of digital inputs on PIC32 would look like this: [6]

```
#include <plib.h>
int main(void) {
      // Set analog pins to digital I/O mode
      AD1PCFG = 0xFFFF;
      // Set pin A9 as digital input
      TRISAbits.TRISA2 = 1;
      // Read from pin value bits
      A2_value = TRISAbits.RA2;
      return 0;
}
```

**3. Connecting to a wireless network based on provided credentials.**

Making the wireless module connect to a wireless network is basically sending a AT command (AT+CWJAP) to the wireless module through UART.

The command has two parameters: ssid and pwd, which corresponds to the SSID and password of the wireless network. The wireless module would respond with OK upon connection success. [7]

The wireless module should also be set to AP mode using the AT+CWMODE command. The command has one parameter: mode. 1 represents station mode, where the wireless module connects to an existing network; 2 represents AP mode, where the wireless module creates its own network; 3 represents AP+Station mode, where the wireless module connects to an existing network and creates its own network which has the same credentials as the existing network. The wireless module works as a wireless extender in this mode. [7]

Pseudocode of connecting to a wireless network would look like this: [8]

```
void esp8266_stationinit(ssid, pwd) {
      // 1: Station mode in AT command
      esp8266_modesel(1);
      // Connect to wireless network under station mode
```

```
      esp8266_stationconnect(ssid, pwd);
}
```

**4. Creating a wireless network.**

Making the wireless module connect to a wireless network is basically sending an AT command (AT+CWSAP) to the wireless module through UART.

The command has three parameters: ssid, pwd, chl, ecn. ssid is the SSID of the wireless network created, pwd is the password of the created network, chl is the channel ID of the created network, and ecn is the encryption method of the created network. (No encryption, WPA/WPA2 or both) The wireless module would respond with OK upon creation success. [7]

For data transmitting to properly function, the wireless module also should set its own MAC and IP addresses for the device to properly recognize the wireless module when connected to its wireless network. Two AT commands, AT+CIPAPMAC and AT+CIPAP, set the MAC address and IP address of the wireless module. [7]

The wireless module should be set to AP mode using AT+CWMODE command as mentioned in earlier sections.

Pseudocode of creating a wireless network would look like this: [8]

```
void esp8266_softapinit(mac, ip, ssid, pwd, chl, ecn) {
      esp8266_modesel(2);  // 2 for soft AP mode in AT command
      esp8266_setmac(mac); // Set MAC address
      esp8266_setip(ip); // Set IP address
      esp8266_softapcreate(ssid, pwd, chl, ecn); // Create wireless network
under softAP mode
}
```

**5. Fetching data from devices through the wireless network.**

The wireless module transmits data to the MCU based on interrupts, that means when the wireless module has transmitted the data (RX FIFO is not empty), it will trigger a system interrupt. In this case, we would handle the data packets in the interrupt handler.

The pseudocode for the interrupt handler of UART1 would look like this: [5]

```
void __ISR(_UART1_VECTOR, IPL1SOFT) IntUart1Handler(void)
{
  // Is this an RX interrupt?
  if (INTGetFlag(INT_SOURCE_UART_RX(UART1)))
    {
     unsigned char * data = NULL;

      // Get the data pointer to the data received.
```

```
      data = UARTGetDataByte(UART1);

       // Clear the RX interrupt Flag
       INTClearFlag(INT_SOURCE_UART_RX(UART1));

       // Handle data correspondingly
       // The data could come from different sources in different modes,
therefore the operation mode should be specified for data processing.
       datahandler(mode, data);
     }

  // TX interrupt is not used here.
  if ( INTGetFlag(INT_SOURCE_UART_TX(UART1)) )
    {
      INTClearFlag(INT_SOURCE_UART_TX(UART1));
    }
}
```

**6. Fail-safe mode implementation.**

Upon several user-defined or built-in emergency conditions, the MCU software should automatically enter a fail-safe mode, where the MCU closes the window and lowers the shades. This could be done by triggering a general exception and controlling the shade motor and the window actuator to close and resetting the MCU software. [9]

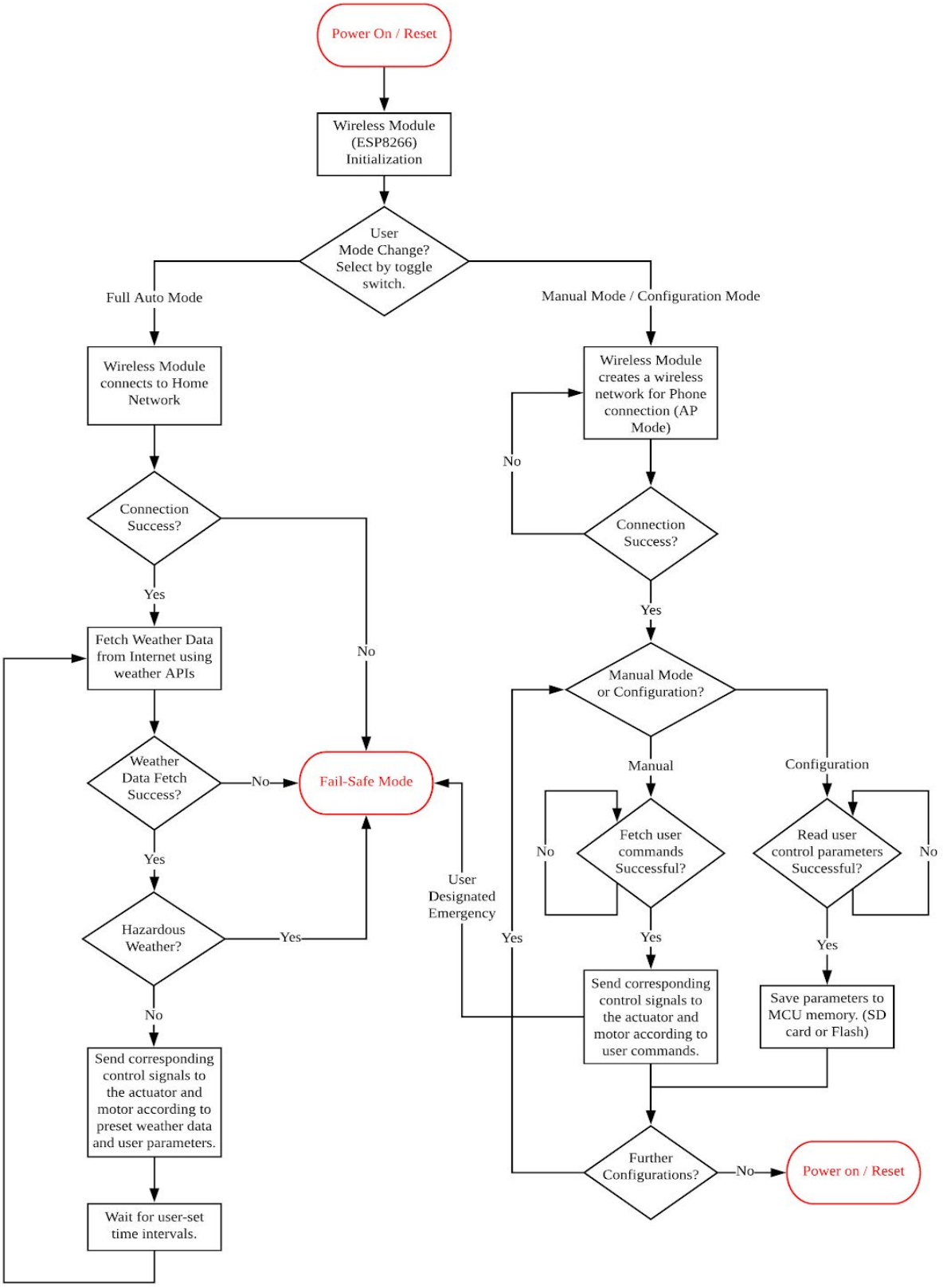
Pseudocode for fail-safe mode implementation would look like this:

```
void __attribute__((nomips16)) _general_exception_handler(void)
{
    // Disable Interrupts
    _CP0_SET_STATUS(_CP0_GET_STATUS()&0xFFFFFFE);

    // Lower windows and shades
    lower_windows();
    lower_shades();

    // Software reset using PIC32 libraries
    SoftReset();
}
```

Here is the operation flow chart of the MCU software:

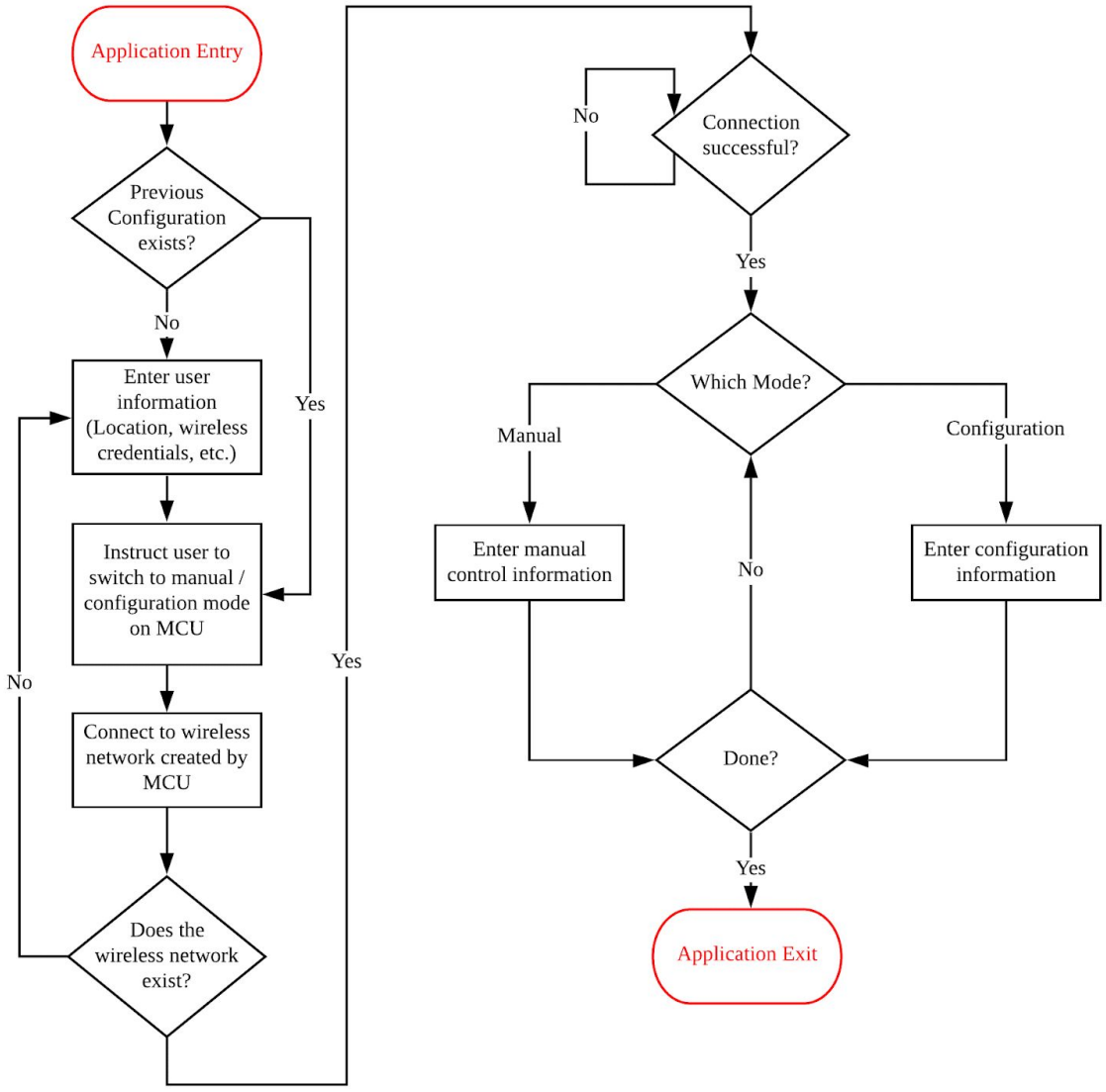


### 2.1.2 Android Application

The Android Application serves a series of functionalities:

1. The Android Application allows the user to enter information, including location and wireless credentials for the local network.
2. The Android Application connects to the MCU through the wireless network generated by the MCU.
3. The Android Application allows the user to either manually control the shades motor and the window actuator (Manual mode) or adjust control parameters that affect the automatic controls in Auto mode.

Here is the operation flow chart for the Android Application:



Here is the list of the implementations of some of the critical software operations in the Android Application.

1. **Scan and check whether the MCU wireless network exists.**
   Android uses a API class called WifiManager for managing wireless networks, including scanning and connecting. [10]
   The code first checks whether Wi-Fi is enabled on the user's phone. Then, it calls an API function to scan for the list of wireless networks. Then, the code checks whether the Wi-Fi scan results contain the SSID of the MCU.

Pseudocode for scanning wireless network would look like this: [11]

```
private void scanwifi() {
      // Check whether Wifi is enabled, if not turn on wifi
      if (!wifiManager.isWifiEnabled()) {
            wifiManager.setWifiEnabled(true);
      }
// Start scanning for wireless networks.
wifiManager.startScan();
// Get Scan results
results = wifiManger.getScanResults();
// Check whether MCU wireless network exists
checkresults(results, mcucredentials);
}
```

2. **Sending data packets over a wireless network to a designated IP address.**
   Android utilizes network sockets as a formal way to send packets over wireless networks. In this implementation, the Android Application and the MCU board will communicate using UDP sockets (Datagram Socket on Android). The Android Application would initialize a Datagram socket and listen on the socket.

Pseudocode for opening a Datagram socket and listening on a socket would look like this: [12]

```
// Opening a socket
private void opensocket() {
// Open a datagram socket on specified port
DatagramSocket UDPSocket = new DatagramSocket(port);
}
// Listening on a socket
private void listensocket() {
boolean run = true;
while (run) {
    try {
        // Generate test packets
        byte[] message = new byte[8000];
        DatagramPacket packet = new
```

```
DatagramPacket(message,message.length);
        // Set timeout and receive packet
        udpSocket.setSoTimeout(10000);
        udpSocket.receive(packet);
        String text = new String(message, 0, p.getLength());
        // Monitor exceptions
        } catch (IOException e) {
            run = false;
            udpSocket.close();
        } catch (SocketTimeoutException e) {
            run = false;
            udpSocket.close();
        }
    }

}
```

### 2.1.3 Circuit Schematic and PCB Board

The circuit schematic for this project is composed of 3 parts: the power supply circuit, the digital circuit including MCU and WIFI module, and the analog circuit containing a H-bridge motor driver. The schematics for each part are presented in the figures below:

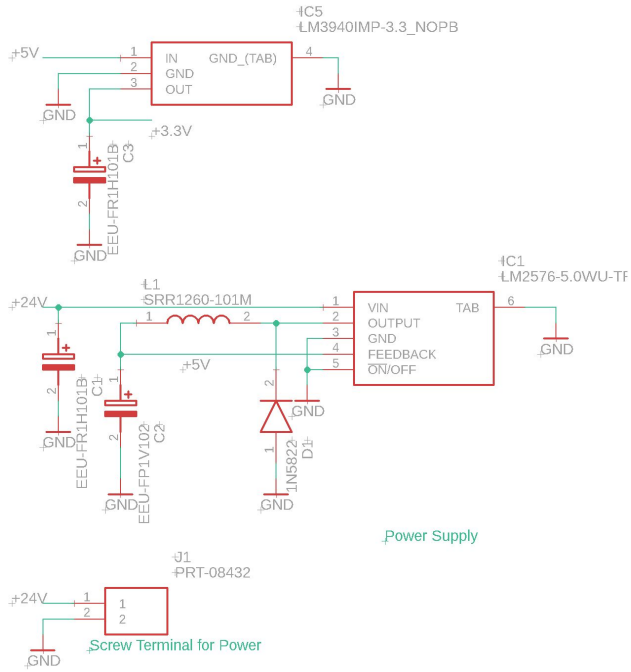



Figure: Power supply circuit schematic

The power supply circuit is composed of a screw terminal, a LM2576-5.0 IC and a LM3940 IC with surrounding circuits. The screw terminal is connected to a 24VDC switching power supply, capable of delivering 50W to the whole system. The LM2576-5.0 would step down from 24VDC to 5VDC, delivering up to 3.0A and powering the digital circuit as well as the internal logic translation in L293DNE quadruple H-bridge IC which would consume up to 1.0A at 5VDC. The LM3940 would convert 5VDC to 3.3VDC with current up to 1.0A, sufficient for power consumption of the MCU and WIFI module.

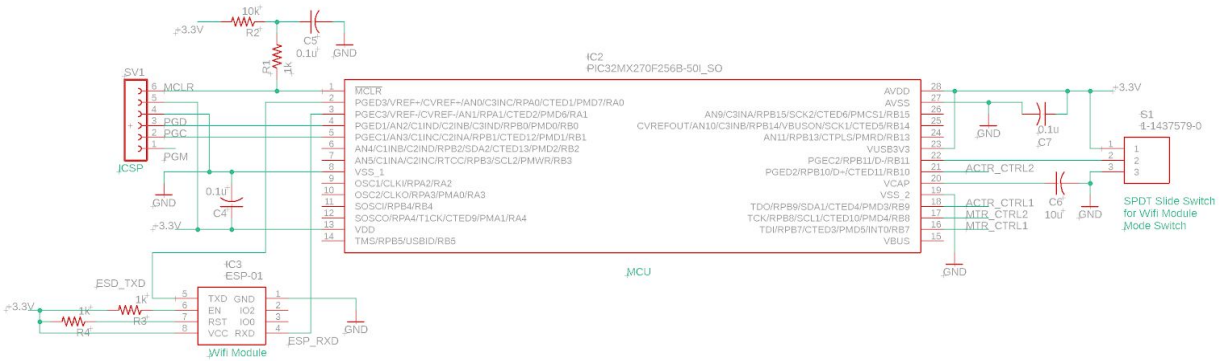



Figure: MCU and WIFI module schematic

The MCU schematic contains the required surrounding resistors and capacitors defined in the datasheet. A 6-pin header is connected to the MCU for ICSP in-circuit programming purposes. An SPDT slide switch is connected for the user to switch the mode of the WIFI module between AP (Access Point) Mode and STA (Station) Mode. The ESP8266 ESP-01 WIFI module is connected with MCU for UART communication.

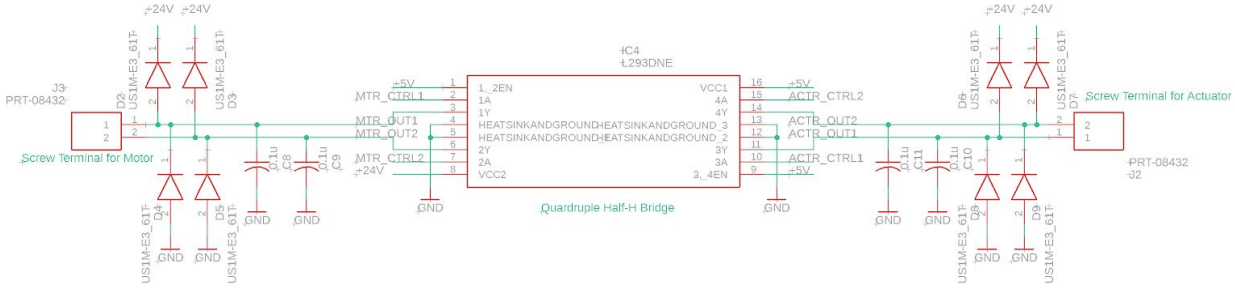



Figure: Motor driver circuit schematic

The motor driver circuit is composed of a L293DNE quadruple half H-bridge capable of drawing 1.0A per channel from the 24V power supply. Flyback diodes are connected between motor and 24VDC source as well as motor and ground to eliminate voltage spikes across the inductive load.

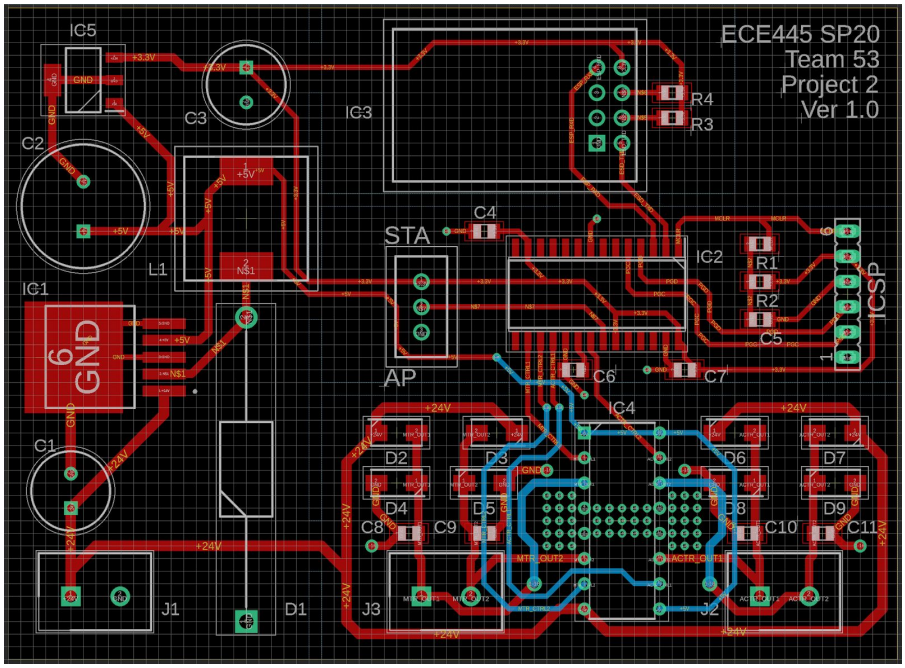



Figure: PCB board layout in Autodesk Eagle

The figure above presents the PCB board layout in Autodesk Eagle. The circuit board is composed of 4 parts: power, analog, digital and RF parts. The circuit design would handle at most 2A at 24VDC. Using the online PCB trace width tool[13], setting the current to 2.0A and thickness to 2oz/ft^2, the result of 40mils trace width is obtained. For the +24V traces, the trace width is set to 40mils. For other traces in analog circuits, trace width is set to 32mils. For digital circuit and RF circuit, trace width is set to 20mils. The ground plane at the bottom layer of the PCB is divided into 4 parts, as presented in the PCB layer view below, extracted by online gerber RS274x viewer [14]:

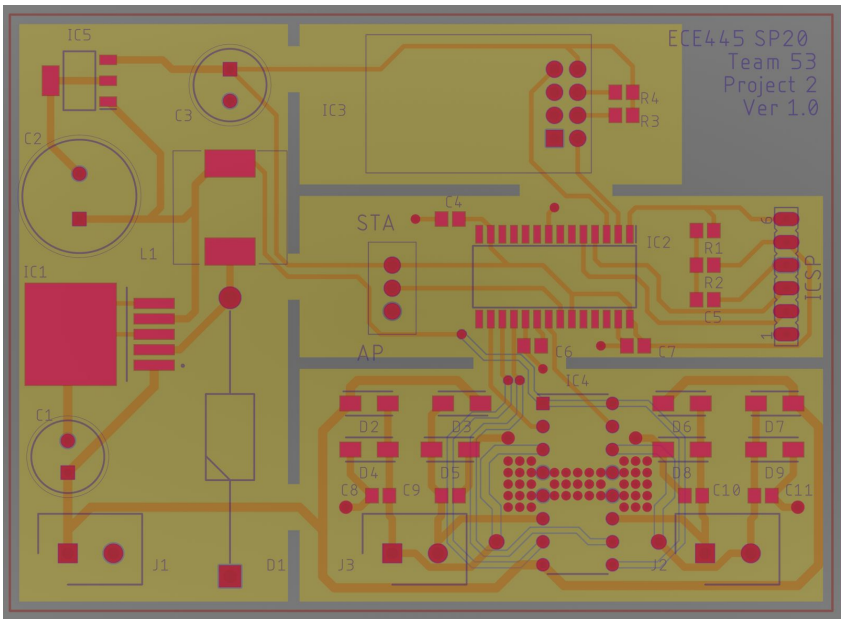



Figure: PCB Gerber RS274x file layer view

The left part is the ground plane for the power circuit. The top middle part is the RF circuit, which carries the WIFI module. The right middle part is the digital circuit for the MCU and the right bottom part is the analog circuit for the motor driver. Under and surrounding the IC4 silkscreen are 45 GND vias for heat dissipation from the ground pins of L293DNE to the ground plane. The three RF, digital and analog ground planes are bridged by a copper trace of 200mils with the power ground plane. Between the RF and digital, and digital and analog ground planes, bridges of 400mils of copper are reserved for the signal traces and return traces to run over for the communication between subcircuits. By properly separating the grounds, electromagnetic interference (EMI) is reduced.

### 2.1.4 Partial Circuit Simulation

The PSPICE models of LM2578-5.0BT and L293D are available from Texas Instrument. The simulation for each of these two components are achieved in Cadence Capture CIS with Cadence PSPICE. The schematic and simulation results are shown below:

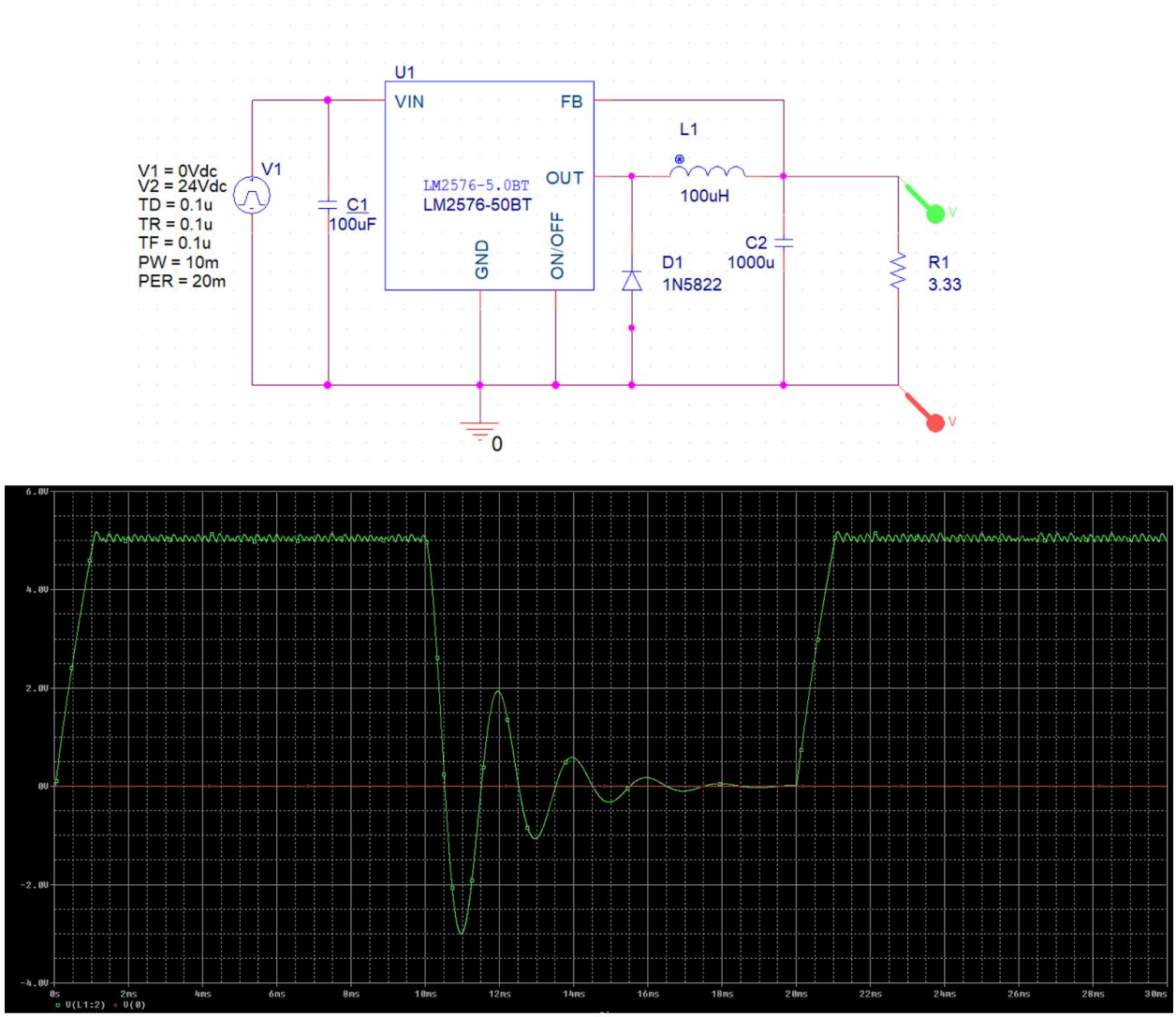



Figure: LM2576-5.0BT schematic and transient simulation

As shown in the graph, after the power is turned on to 24VDC, the 5.0VDC output of LM2576 is able to stay stable after 1ms. The ripple of the output, when a resistive load is consuming 1.5A, is below 5%. When the 24VDC is turned off, the undershoot would be stabilized in about 10ms.

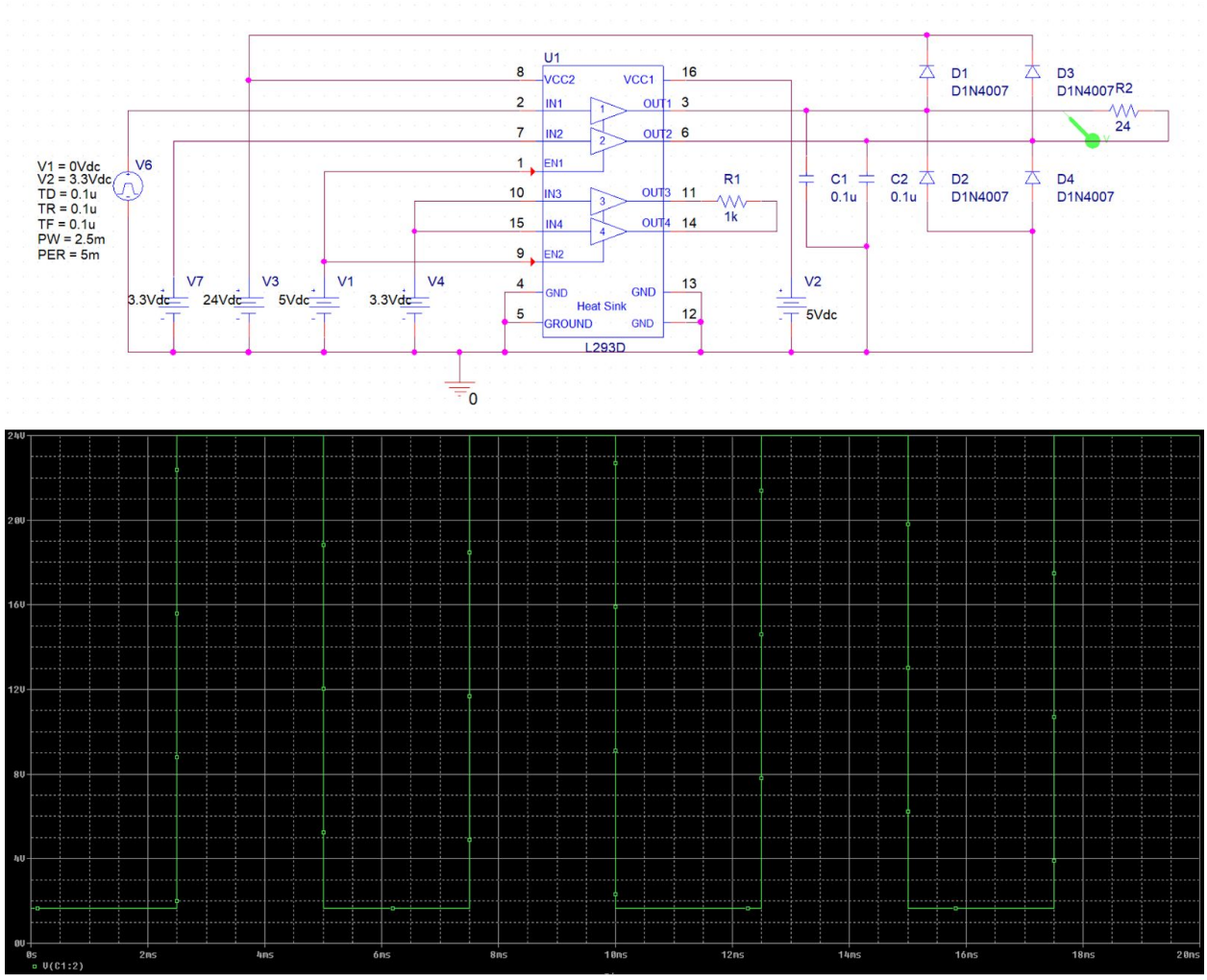



Figure: L293D schematic and transient simulation

The transient simulation of L293D above shows that the L293D IC would work with a high logic voltage of 3.3V under that the internal logic translation is performed at 5V. The input of a 200Hz 50% duty cycle 3.3V squarewave with 0V offset into IN1 pin with a constant 3.3Vdc into IN2 pin would create a perfect 200Hz 50% duty cycle 24V squarewave at the output with a load current of 1A.

# 3. Second Project Conclusions

## 3.1 Implementation Summary

Canlin was able to implement some of the control code for the wireless module (ESP8266), including initialization, wireless connections (AP/Station Mode) and handling of data packets. He also was able to implement the code for initializing the UART interface. Canlin also searched on the Android API for wireless network, network sockets and UDP sockets. Kaiwen was able to finish the first version of the PCB design and simulate two of the ICs used in this project. Although the labs are closed and parts are not available, we succeed in prototyping the software and hardware of our project. While Yihao was not able to come up with the modeling of the circuit, he planned on some procedures that can be possibly done in testing once the lab access is available again.

## 3.2 Unknowns, uncertainties, testing needed

Because of the delay in logistics and labs are closed, the tests and measurements of the power and motor driver subsystem are unable to be carried out. Due to the lack of physical PIC 32 MCU board and ESP8266 wireless module, we were not able to test the MCU software. Also, due to lack of time for project 2 implementation, we were not able to come up with a complete Android App, even at a prototype level. Once the lockdown is lifted, we would be able to access the lab and unit test the subsystems of our design. Also, when given enough time, we would be able to implement the Android Application and test its functionality.

## 3.3 Ethics and Safety

There are few safety concerns in our project. The first one is the power conversion safety issue. The AC-DC converter has an overvoltage CAT III rating which needs to be handled carefully. The power supply with power rating of 50W and 24V falls under the Class 2 of NEC standard, requiring considerations including the location, humidity and being properly grounded[15]. A protective case can be added to enclose our circuit and prevent any short-circuit hazards.

The second concern is the malfunction of the mechanical subsystem. This can happen due to the power shutdown inside the building. However, this event is likely caused by a storm or hurricane, and which will be predicated by the online weather data. So the control system will make decisions ahead of time and close the window to prevent flooding.

The third concern is information safety. For the control subsystem to process weather data, location information is needed. The user enters the location information into the user application, and location information is sent to the control subsystem through wireless network. The location information contains the exact address of the user, traceable to the exact street name, zip code, city and state, and the longitude/latitude information. This is sensitive information which, if fetched by attackers, might be used for malicious purposes. To ensure information safety, the source of information must be secure, and the life-time of sensitive information should be kept as short as possible. The user application will delete the location information right after it is sent to the control subsystem, and the location information will be gathered from Android location services API, which provides adequate security. [16]

Since our design will avoid all the sensors in existing projects, there will be no short circuit caused by exposing any sensor outside the window. With the change of the design and the disclosure of any safety issues stated above, we therefore uphold the IEEE Code of Ethics #1[17].

## 3.4 Project Improvements

Since our second project relies on fetching data online and processing these data to allow adjustments on the windows, we can further upgrade this project by integration into the available smart home solutions, such as Apple Homekit, Google Assistance, and Amazon Alexa. This way will allow seamless collaborations between our weather-adaptive window and air-conditioning unit for better user experiences and potentially saving even more money from electricity bills. The process toward this improvement could be very time-consuming since we have to consider multi-platform softwares.

In dealing with the situation where a power outage occurs, a battery and a boost converter can be employed in our design to ensure the window is still fully functional and give the user enough time to make adjustments on the windows. This update is an extra measure to avoid safety issues caused by sudden power outages.

To reduce the risk of CO gas poisoning in some occasions like the homeowner forgetting to turn off the oven after cooking, connecting the smoke and CO detector to the window system wirelessly can be effective. In this case, the window can be quickly adjusted to open and improve ventilation while the user is trying to cut off the power for the oven.

# 4. References


[1] WhaTech, 'Recent research: Global Window Automation market worth US$ 7970 mn at 6% CAGR forecast by 2025', 2019. [Online]. Available: https://www.whatech.com/market-research/consumer/588419-global-window-automation-market-worth-us-7970-mn-at-6-cagr-forecast-by-2025-supply-demand-market-research. [Accessed: 03-Apr-2020]

[2] "How Reliable Are Weather Forecasts?", SciJinks, [Online]. Available https://scijinks.gov/forecast-reliability/ [Accessed: 17-Apr-2020]

[3] Leah M, "Long Term Expanding-Disk Rain Sensor Accuracy", *ASCE Library*. [Online]. Available: https://ascelibrary.org/doi/10.1061/%28ASCE%29IR.1943-4774.0000381 [Accessed: 17-Apr-2020]

[4] phaetto, "PIC32-ESP8266-WifiExample", Github. [Online]. Available: https://github.com/phaetto/PIC32-ESP8266-WifiExample [Accessed: 6-May-2020]

[5] "Peripheral Library Code Examples", Cornell University. [Online]. Available: https://people.ece.cornell.edu/land/courses/ece4760/PIC32/PLIB_examples/plib_examples/ [Accessed: 6-May-2020].

[6] Andrew Long, "PIC32MX: Digital Inputs", Northwestern University. [Online]. Available: http://hades.mech.northwestern.edu/index.php/PIC32MX:_Digital_Inputs [Accessed: 6-May-2020].

[7] espressif, "ESP8266_AT", Github. [Online]. Available: https://github.com/espressif/ESP8266_AT/wiki [Accessed: 6-May-2020]

[8] camilstaps, "ESP8266_PIC", Github. [Online]. Available: https://github.com/camilstaps/ESP8266_PIC [Accessed: 6-May-2020]

[9] "PIC32MX Interrupt and Exception Usage", Microchip Inc. [Online]. Available: https://microchipdeveloper.com/32bit:mx-arch-exceptions-usage [Accessed: 6-May-2020]

[10] "WifiManager", Android Developer Documentation. [Online]. Available https://developer.android.com/reference/android/net/wifi/WifiManager [Accessed: 7-May-2020]

[11] Sylvain Saurel, "Develop a WiFi Scanner for Android", Medium. [Online]. Available: https://medium.com/@ssaurel/develop-a-wifi-scanner-android-application-daa3b77feb73 [Accessed: 7-May-2020]

[12] “Android UDP Client Example”, msdalp. [Online]. Available: https://msdalp.github.io/2014/03/09/Udp-on-Android/ [Accessed: 7-May-2020]

[13] “PCB Trace Width Calculator”, Advanced Circuits. [Online]. Available: https://www.4pcb.com/trace-width-calculator.html [Accessed: 7-May-2020]

[14] “Gerberlook”, Pad2Pad. [Online]. Available: https://www.gerblook.org/ [Accessed: 7-May-2020]

[15] Industrial Automation & Controls, ‘Technical Note NEC Class 2 DC Power Supplies Revision 2 / May 3, 2002’, 2002. [Online]. Available: https://escventura.com/manuals/sola_introNECclass2_rg.pdf. [Accessed: 3-Apr-2020]

[16] Danny G., *Android Developer’s Guide to the Google Location Services API*, 2015. [Online]. Available: https://www.toptal.com/android/android-developers-guide-to-google-location-services-api [Accessed 03-Apr-2020]

[17] Ieee.org, "IEEE IEEE Code of Ethics", 2016. [Online]. Available: http://www.ieee.org/about/corporate/governance/p7-8.html. [Accessed: 14- Apr- 2020].