Senior Design 2 Project Documentation

Auto FBO

University of Central Florida
College of Engineering and Computer Science

Dr. Lei Wei & Mr. Michael Young

Group 12

Joshua Dean, EE Michael Graziano, EE Vanessa Pena, CE Gilbert Vieux, CE

Table of Contents

1. Executive Summary

Before a pilot takes off they should know that their microphone, radio, and headset are operational. This is so that they can communicate with other pilots in the area to avoid deadly collisions and for communicating with Air Traffic Control soon after takeoff. This is necessary if a pilot is planning to fly IFR as they must establish communication with ATC starting on the ground or soon after becoming airborne. Also, as they begin their take off or come into land, a key piece of information is to know the wind direction, wind speed, and gusts at the airport they are taking off or landing at. This is because pilots always need to take off and land into as much as a headwind to increase the amount of wind over the wings to generate lift, increase airspeed, and decrease groundspeed. Current wind information is crucial especially if a crosswind exists, as the pilot needs to choose the best runway to take off or land on. Other weather information such as temperature and the barometric pressure at the airport is also important so that pilots can set their altimeters and judge the density altitude as well as the visibility.

Usually an Automated Weather Observing System (ASOS) or an Automatic Terminal Information Service (ATIS) and FBOs are the ones to relay this information as well as other remarks about airport conditions to the pilots over the radio, but some airports do not have a FBOs, ASOS, or ATIS. Furthermore, most FBOs are not staffed 24 hours a day throughout the year. One solution to try to mitigate this issue at such airports is a windsock, which is a light and flexible cone of fabric mounted on a mast, usually somewhere along the airstrip of an airport. Windsocks let the pilots know some of the important weather readings, such as wind direction, but they are small and cannot be seen until the aircraft is very close to the airport. On the other hand, there are some automated systems currently on the market that perform task such as broadcasting weather conditions and transmit radio checks, but they are costly and not suited for smaller airports.

The Automated Fixed Base Operator is a low-cost system that satisfies these two basic needs. This system broadcasts important weather information when prompted by pilots in the area. For example, when the system is prompted, the system will broadcast a weather report that includes the latest recorded wind direction and wind speed as well as gusts, temperature, dew point, density altitude, and airport remarks. This system also performs a transmit radio check for any pilot that consists of recording the transmission from the pilot and playing it back along with the power level so the pilot knows exactly

how operational their equipment is. Therefore, this system can be classified as an Automated Fixed Base Operator for small airports. The Automated Fixed Base Operator would act as a hub of communication for these small airports that do not have a dedicated FBO or weather station, as well as FBOs and airports who wish to automate this service fulltime. This system provides a source from which any pilot can obtain crucial weather information or perform any radio communication checks they need prior to taking off and landing their aircrafts.

Our goal in the design of this Auto FBO to connect a weather station and VHF radio through an interface board to a microprocessor that can process all the necessary information required to be comparable to modern ATIS and ASOS systems, as well as preform quality radio communication checks. Using these components, we build a system that can assist pilots in taking off, flying, and landing safely, while being configurable and cost-effective.

2. Project Description

This section describes the motivation, goals, objectives, and some of the key systems of this project to better understand its premise and the features it has. We also detail the issues that this system solves, what causes those issues, and who would benefit most from this system.

2.1 Project Justification and Motivation

The vast majority of airports in the U.S. as well as other parts of the world are non-towered airports. Many towered airports even have non-towered hours of operation, usually during night hours. Non-towered airports or hours of operation is when the air traffic control tower (if there is one) is empty and not staffed. This means that there is no one available for pilots to communicate with besides other pilots. It becomes the pilots sole responsibility to be aware of the current weather conditions, whether or not their radio is operational, and where other aircraft are located. When active, towered airports assume those tasks and are held responsible to maintain safe, orderly, and expeditious flow of air traffic, as well as report accurate and real time weather observations. However, when pilots fly into and out of non-towered airports they are responsible to maintain good communications while operating in the local airspace as well as on the airport's runways and taxiways. Also, the local weather at many non-towered airports is not automatically broadcasted over a local frequency and is usually found from another nearby airport's weather report.

One concern pilots face when preparing to fly out of a non-towered airport is how well their radio is working. It is vital for a pilot preparing their aircraft for flight to ensure that their communications systems are properly working. This is especially true for pilots flying under Instrument Flight Rules (IFR), as they must establish contact with air traffic control soon after becoming airborne. With no tower they can only perform a radio check if there are others on the local frequency, which is never guaranteed.

The current local weather is also a concern for both pilots flying into and out of non-towered airports. For pilots flying out of a non-towered airport getting the current local weather is usually done by looking up the weather, observing outside conditions, and collecting nearby airports weather reports. Pilots flying into a non-towered airport, however, do not have the luxury of looking up the current local weather from their plane. The best a pilot flying into a non-towered airport can do is to lookup the weather they will be traversing through beforehand, observe the windsock at the airport, remain conscious

of weather conditions around the aircraft, and tune into nearby airport's weather reporting stations. At a towered airports this complication is resolved with an Automatic Terminal Information Service (ATIS) or another equivalent system, which provides highly accurate and current weather as well as other remarks (obstructions near the runways, closed taxiways, other weather information, etc).

In respect to weather, pilots are interested in elements such as the wind speed and direction, barometric pressure, temperature, and dew point surrounding the airport when preparing for a flight, taking off, and landing. Wind speed and direction are most important for pilots, because they dictate which runway pilots will use to take off and land. This is because during the takeoff and landing phase it is desired to have as much wind flowing over the wings of the aircraft as possible to increase both drag and lift. Barometric pressure is used to tune the aircraft's altimeter, which indicates the altitude of the aircraft. Lastly, temperature and dew point are used to judge the density of the air and predict the visibility conditions. The temperature along with elevation gives pilots information on how well their aircraft will operate and if their aircraft is safe to operate in the air. The difference between temperature and dew point gives pilots information on the visibility surrounding the airport. This is used decide if an area's airspace is under Visual Flight Rules (VFR) or Instrument Flight Rules (IFR).

Our motivation for this project is to improve the safety of pilots and passengers at these smaller airports with no manned Field Base Operator (FBO). When pilots aren't sure of weather conditions they do not know which runway to land on and if they can't be sure their communications systems are operational, then they run the risk of missing important communications or not being able to transmit their location or intention to other pilots. The airports that don't have a dedicated FBO usually don't have the financial means to fund the expensive automatic weather systems on the market which can run upwards of thousands of dollars. Our system would become the new model for a cost-effective solution and would give hundreds to thousands of airports around thee world access to a previously unattainable lifesaving system.

The proposal for this project was brought by Professor Michael Young last summer to be completed by a senior design group at UCF. Unfortunately, the final product they presented was undeployable and did not satisfy all of Professor Young's needs. We seek to improve on the areas where the previous team fell short; expanding the weather capabilities of the weather reporting system and delivering a "no distortion added" communications check.

2.2 Goals and Objectives

The objective of this project is to build an easy to use, reliable, and efficient system for pilots to receive critical weather information and perform a communications check when flying into a non-towered airport. Our system will provide more information to pilots than a typical windsock which will give them the data they need to be able to take off and land safely. This system will be comparable to the existing Automatic Terminal Information Service (ATIS) and Automated Surface Observing System (ASOS) systems in place at larger airports so that pilots will already know what to expect and not have to learn a whole new protocol.

The system will be able to recognize a mic click signal from the pilot and decide from the signal if the pilot is requesting weather condition information or a communications check. If the pilot is requesting weather information, the system will respond with an ATIS style broadcast with the wind speed, direction, visibility, temperature, humidity, and pressure. If the pilot is requesting a communications check, the system will respond with a message acknowledging the request and will record and playback the pilot's response so they can hear exactly how their message was received. The system will also respond with a power level to inform the pilot of their signal strength.

A similar system was designed for a previous senior design project but that system did not meet all the requirements and was too complicated and cumbersome to deploy. Their audio playback for the communications check was not integrated into the PCB so to receive, save, and playback a pilot's transmission, they had to use a separate USB interface on a computer. This affected the quality of the transmission but it also made the system much bulkier. To deploy their system, they needed room for the weather sensors, PCB and microcontroller, and a separate computer to process the audio. The idea behind the communications check was that it allows the pilot to make sure they can be heard by other pilots or air traffic control towers but this becomes ineffective when the playback is distorted. Their communications check failed because of that crucial factor. Inaccurate playback will cause the pilot to believe their transmissions are worse than they are so they will make unnecessary adjustments furthering the problem.

Our system will differ from the previous senior design project in many key ways. We will be integrating all the components, aside from the weather sensors, onto one chip so that they system is contained and very easy to deploy. This will include a codec to receive, save, and playback a pilot's communications check so that the playback is as accurate as possible and they pilot will also receive a quantified value for the quality of their transmission. In addition to this improved communications check, we will also be including

more weather sensors and more robust logic to allow the pilots to get the most accurate weather information when they request it instead of clogging the line with repeated information. Instead of just reporting wind conditions, the system will also report temperature, humidity, and air pressure. These are all crucial measurements for pilots because it allows them to understand how the wind will affect their plane and what counter measures they will need to take. In addition to these changes, we are also simplifying the circuits immensely. The previous team added many unnecessary components and overcomplicated the circuitry so we started with an all new design and chose to incorporate and build off more out of the box components such as the codec. This way we are able to pull what we need from each component and combine the simplified circuitry into the PCB.

2.2.1 Weather Conditions Report

The weather conditions report is one of the main functions of the system. When the user/pilot keys the mic on their radio a specified number of times, the system should broadcast weather conditions. This weather conditions report should include wind speed accurate within ±2 knots, wind direction within ±5 degrees, temperature within ±3 C, humidity within ±4%, and air pressure within ± 0.0591 inHg. It will also need to check if the channel is occupied and only broadcast the weather report when the channel is unoccupied.

Another feature of this function is to broadcast an updated weather report if the wind conditions change more than a specified amount. For example, if the system broadcast that winds are 5 knots at 120 degrees, and they change to 10 knots or 150 degrees, the system will broadcast the new wind conditions so that the pilot is always up to date with the most current and accurate conditions.

This also touches on the Crosswind Alert the system will have. A crosswind is when winds blow near perpendicular to a runway, and this causes makes landing more difficult. Our system will detect when a crosswind exists and broadcast an alert. The system should also announce when a runway is “favorable” to land on. A pilot wants to land into headwind so the length of their landing is shorter. If the system detects winds are more than, say, 5 knots and they are in the direction of a runway, the system should announce that that runway is favorable to land on.

2.2.2 Transmit Radio Check

The second main function of our system is a Transmit Radio Check. Before a pilot takes

off, they want to ensure that their mic, radio, and headset work so they can communicate with Air Traffic Control (ATC) and other pilots. Normally, the pilot would contact the Field Base Operator (FBO) and the FBO would respond with a radio check and wind conditions. Our system will be used at an airport without an FBO. When the user keys the mic a specified number of times, the system should prompt the user to perform a Transmit Radio Check. The system will record what the pilot transmits, and play it back exactly how it was heard. Then the system will announce the power level of the transmission. This way the pilot can verify their mic and radio are operating normally and that their signal strength is satisfactory. During this process, the system will verify that the channel is not occupied before transmitting the prompt or the recording.

2.2.3 Printed Circuit Board Interface

To interface the handheld radio and the microcomputer we will need to design and build custom circuitry and ultimately fabricate a Printed Circuit Board (PCB). This PCB will have all necessary inputs from the radio and convert them into usable signals for the microcomputer. The PCB will also have these power supplies. In turn, it will also create usable signals for the radio that come from the microcomputer. The weather sensors will also be connected to the PCB and accessed by the microcomputer.

2.2.4 Web Interface

The web interface is intended to provide an easily accessible graphical interface for the user. The interface would provide the user with valuable information concerning the current weather conditions; this includes wind speed, wind direction, gust, and temperature. The interface would allow users to check the current conditions at the airport from anywhere and at any time. The system will also allow the admin user for the airport to switch the click pattern for requesting each task, like a communications check, to best fit their preference and to ensure the click pattern does not conflict with other systems already existing at the airport. The operator would need to switch the click pattern if the current click pattern interferes with any patterns already established at the specific airport because if not then pilots may not be able to perform necessary tasks like turn on runway lights.

Our device will also host a local web server that will provide a graphical user interface that anyone can use to get information from the system. The user will be able to specify any parameter and adjust the system. For example, if the administrator for the system wants to change the number of clicks for the weather report, they will be able to change that from the interface. We also will show a graphic of the runway, a compass overlay,

and the wind conditions so that the user can get a graphical representation of the current weather situation like what is shown in Figure 2.1. The user should be able to type in the IP address or a web address related to the IP of the microcomputer to access the web interface. This system will be opened using the port routing functions of our microcontroller to also allow access from outside of the local network, allowing the user to be able to get weather conditions from an outside location, i.e. their home or office.

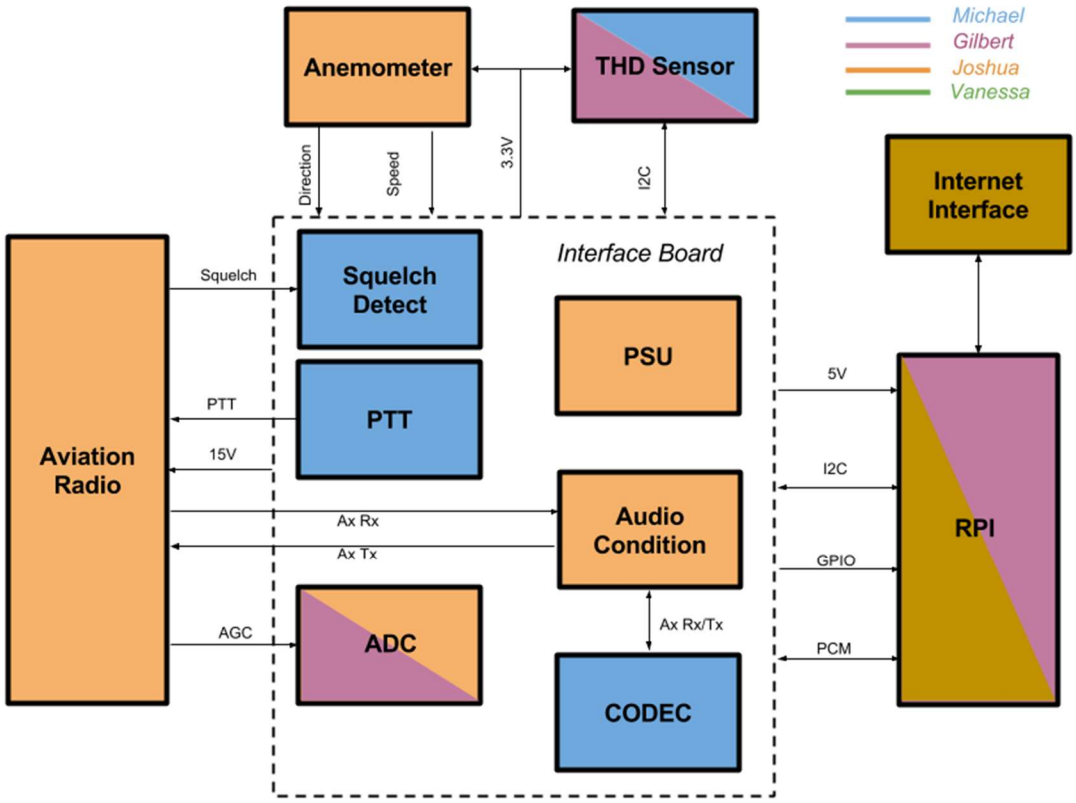


Figure 2.1 General Block Diagram of Auto FBO

2.3 Product Specifications

In this section, we list the specifications to which we believe our system should perform; touching first on the general system specifications such as system size and response time, next we outline exactly how the critical features of weather reporting and communications check should operate and their specifications.

2.3.1 Engineering Specifications

-- Weather Reporting Capabilities: Temperature, Dew Point, Barometric

Pressure, Wind Speed, Wind Direction, Density Altitude
-- The system shall have a web IP graphical interface from which the user can read the current winds and make parameter changes.
-- The system shall operate on the airports UNICOM frequency and shall not broadcast if the radio channel is occupied.
-- Upon receiving the designated cue for a weather report, the device shall return an automated weather message in a precise formatting specific to aviation procedures.
-- The system shall update the pilot and broadcast the current wind conditions if they change such that they exceed the chosen parameters.
-- The system shall announce crosswind and gust warnings if they are present.
--The system shall announce a favorable runway if conditions fall within chosen parameters.
-- Upon receiving the designated radio cue for a communications check, the device shall record the pilot's transmission and subsequently transmit the recording back with no added distortion to the pilot for verification.
-- Following the playback of the recording the device shall transmit a message to the pilot detailing the received message's power level.

Response Time	< 3s
Temperature Accuracy	± 2°C
Humidity Accuracy	± 5%
Pressure Accuracy	± 0.12 inHg
Wind Direction Accuracy	± 5°
Wind Speed Accuracy	± 2 kts or ± 5%
Recording Length	≤ 15s
Power Level Accuracy	± 5 dBm
Playback Correlation	< 0.9

2.3.2 Trade Off Matrix

		Implementation Time	Temp. Accuracy	Humidity Accuracy	Wind Speed/Direction Accuracy	Barometric Pressure Accuracy	Dimensions
		-	+	+	+		-
Good Sound Quality	+	↓ ↓	-	-	-	-	↓
Ease of Installation/Setup	+	↑ ↑	-	-	-	-	↑
Low Cost	-	↑ ↑	↓	↓	↓	↓	↓
Quick Responsiveness	+	↓ ↓	-	-	↓	-	-
Multiple Measurements	+	↓ ↓	↑ ↑	↑ ↑	↑ ↑	↑ ↑	↓ ↓
		< 23 weeks	< ± 3 C	< ± 4%	< ± 2 knts. < ± 5 degrees	< 0.0005 inHg	< 2 ft. on longest side

↑ ↑ *Strong Positive Correlation* ↑ *Positive Correlation* ↓ *Negative Correlation*

↓ ↓ *Strong Negative Correlation* + *Positive Polarity* - *Negative Polarity*

3. Research

This chapter describes existing products both commercially available and the previous Senior Design project for this system. Additionally, the chapter includes the research done for component selection, communication protocols, programming language selection, the various interfaces between components, and a discussion on power supplies.

3.1 Existing Products

Currently there are numerous options when it comes to autonomous or unmanned control tower like services. They typically provide pilots with necessary information like the weather conditions and radio checks similar to what our system will provide. However, these products usually provide way more services for the pilots like monitoring traffic in the surrounding airspace and relaying that information. In addition to the autonomous FBO's, there is also the more traditional approach of having a dedicated FBO at the airport. While these systems share similarities in capabilities they also share a similarity that also happens to be their biggest flaw: having a high cost. Between initial system costs, installation or construction, and routine maintenance or operating; these factors can lead to quite the costly investment in the long run. For some airports, this is a completely justifiable cost, for other small airports this is not the case and will typically lead to the airport being unmanned and unavailable to provide critical information to any pilots.

Figure 3.1.1 Potomac Aviation Micro Tower

The first similar product is the Potomac Aviation Micro Tower (Figure 3.1.1). The Micro Tower is an all-in-one system that operates on the area's CTAF frequency (Usually UNICOM) and provides the same core services that our system will provide. The Micro Tower can broadcast weather conditions, altimetry, visibility, and runway advice. The Micro Tower can also perform the same communications check that our system will have by recording and playing back a pilot's transmission and giving the power level of that received transmission.

Where this system exceeds is its AI capabilities with all that information. For example, the Micro Tower can sit in the radio channel and detect when a new airplane enters the airspace, giving that pilot a greetings and introduction to using the system. Another advantage of the Micro Tower is that it is completely solar powered, meaning it can be set up anywhere in the world and not have to rely on a power source. This leads to an incredibly easy user setup experience; only need two individuals and about a half of a day's work to get the system up and running.

However, airports like Orlando Apopka don't necessarily need or can't afford the multiple thousands of dollars cost of dedicated weather and broadcasting equipment. As mentioned earlier, the Micro Tower fails at being cost accessible for small airports. With a quoted price starting at $75,000, this puts the system in a budget range that is too much for an airport such as Orlando Apopka.

Figure 3.1.2 Unmanned Control Tower

Another similar solution is an unmanned control tower. This is not necessarily a buyable product like the Micro Tower, but should still be considered as a method to compare similarities, usability, and the effectiveness of our system.

These unmanned control towers have the eyes and ears of a standard control tower, with none of the personnel. On average, they can reach heights of 80-feet tall and house high-definition cameras that send the information back to controllers, stationed at a manned ATC Tower. The cameras are spread out to eliminate blind spots and in the future, can be equipped with infrared technology to operate at night or in bad weather.

Overall these solutions again, far exceed the needs of a small airport such as the Orlando Apopka airport, and the price is similarly outlandish when you take into consideration that an airport like Orlando Apopka is mostly self-funded. The Orlando Apopka airport could not afford the expensive Micro Tower and wanted a similar product without the cost, which is why we are building this low-cost solution for them.

Our solution will most importantly be low cost but it will also deliver the functionality that is critical to the safety and efficiency of unmanned airports. We will deliver a easy to use weather reporting system which when requested, inform pilots of the current wind speed, wind direction, temperature, humidity, and pressure. We will also deliver an incredibly accurate communication check system. This system will allow the pilot to request the system to record their transmission and then play it back so the pilot can hear exactly how they will sound to other pilots or air traffic control at other airports.

3.1.1 Previous Senior Design Project

Since our advisor, Professor Michael Young, proposed this project last year for a senior design team, we felt it was important to elaborate on the system they created.

Our project is loosely based on theirs being that the overall premise is the same but there are many key differences which are attributed to their major downfalls and what Professor Young expected from the system. Many of the requirements for the project are requirements he set based on how the system would be use and how important certain aspects would be.

He had two main concerns, audio and the weather playback. At the time of his proposal, his airport was considering what their options were in regards to purchasing a weather reporting system. They came across a few solutions including those listed above but they were all incredibly expensive and over budget for this small airport. Young's proposal was

to build a cost-efficient solution that could do both test radio communications and report the current weather conditions.

Their main issue last year was with the audio playback. They attempted to design the system themselves instead of using a CODEC which ultimately resulting in them scrapping that circuitry and opting for a removeable media type solution. They found a software package that was self-contained and could be quickly deployed so they installed it on a flash drive and had the audio stream through their laptop and the flash drive instead of back through the radio. If this wasn't enough of an issue, the audio quality was also subpar for Young's standards. He wanted a virtually distortion free audio playback which would allow the pilot to accurately hear how they sound to other pilots. Since we opted to follow his advice and use a CODEC we had a much easier time manipulating and storing the audio. Though we had issues with the CODEC we were ultimately able to record and transmit audio through the aviation radio with minimal distortion and we were also able to transmit a power level which gives the pilot quantitative information about the quality of their transmission.

Another issue Mr. Young had with the previous project was the weather readings. The previous team only reported wind speed and wind direction. Any pilot can get that information from the wind sock at the end of the runway. What the pilot really needs is to be able to combine the wind information with information on the current temperature, pressure, and humidity, so they can have a better idea of the weather conditions surrounding the aircraft.

To resolve this complaint from Mr. Young, we added an additional sensor with the ability to read temperature, pressure, and humidity. This give the pilot a more complete picture of what conditions around the aircraft are like and they allow the pilot to make quick calculations and observations to help them take off or land their aircraft.

The previous team also did not take into consideration the real-life application and use of this system. They failed to keep in mind practicality and circuit protection. Very little of the circuitry they created was useable because we switched some of their key parts but also because of the lack of conforming to industry standards. Forcing the user to have a laptop available in order to process the audio is not a feasible solution and was not at all what Professor Young had intended. He wanted something completely hand held and designed in a way that the only thing that needed to be mounted was the anemometer which would be attached to the roof our outside wall of the hangar. This way all of the sensors could get the correct measurement.

3.2 Main Control Unit

This section details the options that have been assessed for the main control unit of the system and why the specific system was selected. It also describes the communication protocols, how the various components of the system will communicate, and the language chosen to write the software for the system.

The main control unit of a system receives and sends data that direct the operations of a computer's processor. The MCU translates input information into control signals that are sent to and carried out by the central processor. Using the information obtained, the processor can then communicate accordingly with any attached external device. In our project's case, our MCU receives digital signals (that are first converted by an ADC from analog signals) as input. The input information is then used by our program to output the related information back to the user. The MCU is necessary to communicate between devices providing multiple functions that allows its user to send, receive, and manipulate control signals from other computer devices.

3.2.1 MCU Options and Selection

This section details the two microcontrollers we deliberated over, their specifications, strength, and ultimately the one we chose that best fitted our project specifications. The reason for choosing one microcontroller over the other is also due to their different coding environment and language. Additionally, we also decided to favor the microcontroller the members of our team are most accustomed to the Raspberry Pi.

3.2.1.1 Raspberry Pi 3 Model B

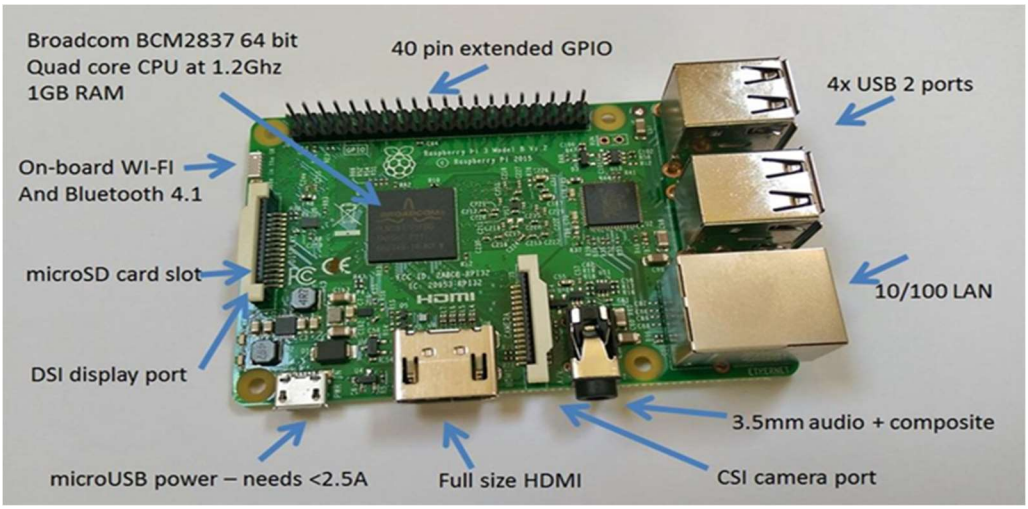


Figure 3.2.1.1 Raspberry Pi 3 Model B Configuration

The Raspberry Pi 3 Model B is a microcomputer equipped with a quad-core 64-bit ARM Cortex A53 running at 1.2 GHz with 1GB of LPDR2-900 SDRAM. This model contains 2.4GHz 802.11n Wireless LAN, Bluetooth 4.1, and 10/100 Ethernet connection. Furthermore, this MCU includes an HDMI port, display interface (DSI), micro-SD card slot for storage, 4 USB ports, and a 3.5mm audio jack. The Raspberry Pi meshes best with the free operating system Raspbian. Raspbian is an optimized distribution of Linux tailored for the Pi. The system provides many packages and pre-compiled software that make the Pi versatile and easy to operate; yet, the Pi's most powerful tool is its GPIO pins. With a total of 40 pins (26 GPIO pins with the rest being power, ground, or I^2C pins), the Pi can communicate and interface tremendously well with external devices.

3.2.1.2 Arduino Uno

The Arduino Uno is a microcontroller that operates at 5V and runs at 16-MHz. The board is populated by fourteen digital input and output pins and six analog input pins.

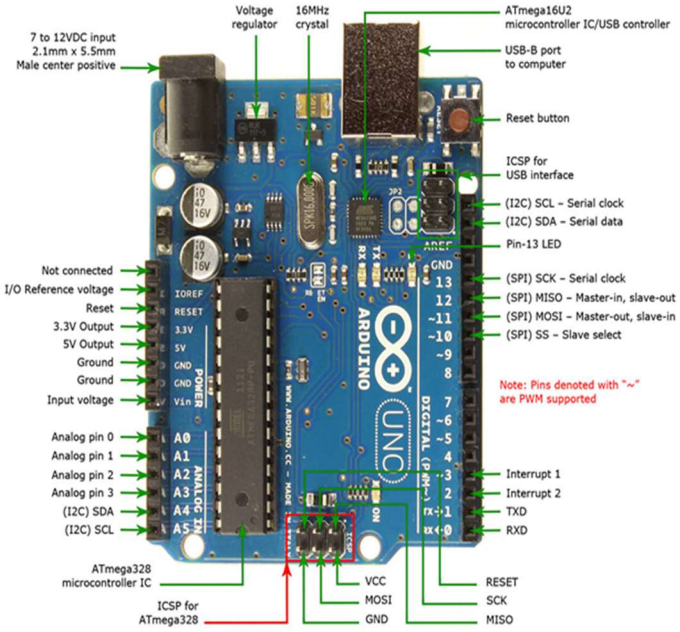


Figure 3.2.1.2 Arduino Uno Configuration

The Arduino Uno's 8-bit AVR RISC-like microcontroller is called ATmega328P and provides 32 kB of flash memory with .5 KB used by the bootloader; it also provides 2-KB of SRAM and 1 kB of EEPROM. Other features include the 32 general purpose registers, an SPI serial port, serial programmable USART; and most conveniently, an onboard 8 channel 10-bit A/D converter. The A/D converter is a required component for our project

since the analog signals from the weather sensors need to be converted to digital signals. The digital signals can then be received and manipulated in order to accurately output the correct response for the weather conditions to the user.

3.2.1.3 MCU Selection

The Raspberry Pi was the clear winner for our project; the Pi was favored not only because of its specifications, but also because the team members had more experience with this specific microcomputer. We researched both microprocessors thoroughly before finalizing our decision; we chose the Pi because of its versatility, accessibility, and open-source libraries. One slight problem was that the Pi lacks an analog-to-digital converter which is needed to process the incoming analog signals from our sensors; on the other hand, the Arduino has a built-in A/D converter while the Pi isn't naturally equipped with one; but, that did not really impact our decision as much because we made use of an external A/D converter paired with our MCU. Figure 3.2.1.3 illustrates the specification differences between the Raspberry Pi and Arduino Uno that are further discussed below.

One of the reasons we selected the Pi is because of its naturally optimized operating system called Raspbian. This Linux-like operating system is distributed with over 35,000 packages and pre-compiled software bundle meant to improve the Pi. It also makes its overall installation process as well as interfacing with peripheral devices quite easy. The programming experience is made simpler by providing a graphical interface to the user. Raspbian is a fully-fledged Linux-based operating system used by the Pi (which in turn is basically a small computer) as stated above, but the Arduino Uno is only a microcontroller. Using the Raspberry Pi 3 as a basic Linux computer allows us to possibly set up a graphical interface in the future, while also providing us with a headless command setup now. The Arduino Uno still supports many functions required by our project. This includes the key function of receiving and converting inputs from sources such as a temperature sensor or anemometer using its built-in A/D converter. Unfortunately, it also does not support a multitude of specifications required by our project such as Wi-Fi access or python.

The Arduino Uno does not provide the user with a variety of coding languages. IDLE's are not compatible (as shown in figure *3.2.1.3*) with Arduino; instead, the user is provided with specifically designed tools to setup and program the different Arduino models. The codes written on the board are known as sketches and are written in C++. This was one of the main deal breakers that pushed our decision towards the more favorable Raspberry Pi. We selected python as our coding language for the ability to interact with Django -a

database framework that allows us to store data on the Pi. Also, python offers many packages to deal with analog signals which further narrowed down our choice of coding languages.

Furthermore, the Raspberry Pi includes a faster processor (running at 2.4 GHz), multi-tasking power (as opposed to Arduino's focus on running one simple program), and it is an independent computer (Arduino Uno is not). The onboard Ethernet network card, the wireless capability, and the graphical interface provided by the Pi shows its superiority with software applications and usability. This graphical interface is an imperative requirement as our sponsor mentioned his desire to change some of the functionalities implemented by our project; such as, changing the current airport location easily or the click-pattern. Also, access to the internet via Wireless Lan or Ethernet connection is required to communicate to our web interface.

Another feature on the Raspberry Pi 3 that contributed to its selection is the 2.4GHz 802.11n wireless capabilities and the 10/100 Ethernet port. This allows us to easily install new software and packages directly from a webpage (as long as there's an internet connection) and set up a local web server. One of the goals of this project is to have a web interface that the user can modify parameters from. Having the Ethernet port lets the user plug in their computer and access a web interface we set up that's run on the Raspberry Pi 3.

Component	**Raspberry Pi 3**	**Arduino Uno**
Model	Model B	R3
Price Range	$35	$22
Dimensions	85 x 56 mm	74.8 x 53.3 mm
CPU	ARM Cortex A53	ATmega328P
Clock Speed	900MHz	16MHz
RAM	1GB	2KB
Flash	Micro-SD card	32KB
EEPROM	N/A	1KB

Input Voltage	5V	7-12V
Min Power	3.5W	.3W
GPIO Pins	26	14
Analog Input	N/A	`8 10-bit
I2C	2	2
SPI	1	1
Dev IDE	IDLE	Arduino Tool
Wi-Fi	2.4GHz 802.11n	N/A
Ethernet	10/100	N/A
USB Master	4	1
Video Out	HDMI, Composite	N/A
Audio Out	HDMI, Analog	N/A

Figure 3.2.1.3 Raspberry Pi Vs Arduino Uno Specs

The 26 GPIO pins on the Raspberry pi was more than enough to finalize our decision. One of the reasons we chose the Pi is because of all the available general purpose pins at its disposal. This variety of pins allows us to interface with our microcontroller and have several pins leftover for backup use. Since the Pi does not have a built-in analog-to-digital converter, we needed to acquire an external ADC converter. We chose the ADS1015 ADC because it fitted our needs and provided more bit precision and power needed by our project.

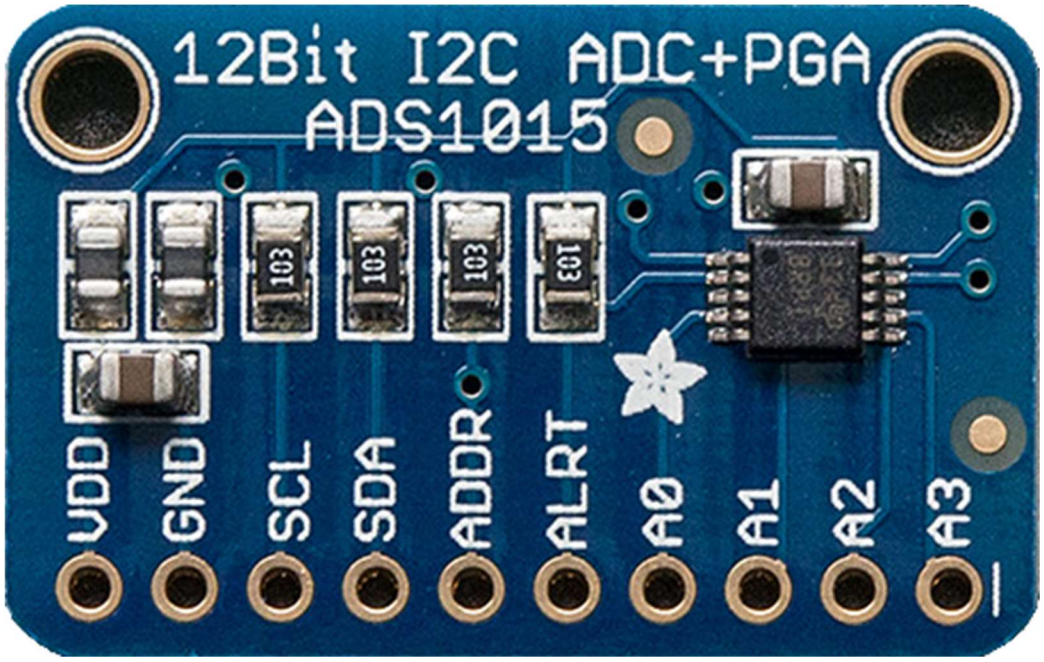


Figure 3.2.1.3.2 ADS1015 external analog-to-digital converter

The ADS1015 is an analog-to-digital converter that utilizes 12 bits of precision to accurately detail the analog signal collected from our sensors. The Pi's accessibility, processing power, multi-tasking capability, and functionalities make it a perfect choice for our project. Also, the I2C bus pins of the Pi meshes quite perfectly with the with the analog-to-digital converter. The I^2C interface also provides a neater wiring between the Pi, ADC, and sensors instead of the way the SPI is configured when wired with the Arduino Uno or the Pi.

3.2.2 Communication Protocols

The nature of VHF Radios in aircraft communication has become critical in the communication of information between traffic control towers and aircrafts all around the country. Radios have communication protocols that need to be addressed prior, during and after communications. These protocols dictate who communicates, which signal propagates in the given frequency band and if your VHF will listen or transmit. These signals will need to be filtered and manipulated in such a way that the Raspberry Pi 3 will be able to interpret them and use them to follow adequate protocol for communication.

3.2.2.1 RX Signal

The received signal from the IC-2A Radio will be sent to the interface board from the positive end of the volume potentiometer. This way we get a clear unattenuated audio

signal from the radio. The importance of this signal is that it will allow the Raspberry Pi 3 to record and save the pilot's communications check audio.

3.2.2.2 TX Signal

The other function of our audio path is to transmit the audio signal from our Raspberry Pi. The TX signal is signal that is send out and carries the transmitted message. During transmission, the half-duplex system will by nature be unable to receive any kind of transmissions.

3.2.2.3 Carrier Detect

Carrier Detect, in communications, is present in the squelch circuit with the function of suppressing the audio output of a receiver in the absence of a higher amplitude and strong input audio signal. The squelch can be opened, allowing all audio signals entering the receiver tap to be heard. This circuit can be useful when attempting to hear weak or distant audio signals. Squelch operates alone on the detection of the strength of the signal; when a device is set to mute, there is no audio signal present. Knowing if there is a carrier detect present, at the squelch, will allow the MCU know when there an audio signal present. We will use the squelch voltage to register when "clicks" have been made by a pilot.

3.2.2.4 Push-to-Talk

PTT has been a standard of two-way radio communication for quite some time. The nature of half-duplex communication systems is that there must be some sort of signal flag to alert the transceiver that it is time to stop receiving and ready for transmission. The reason it is called push to talk is that the action required for this stage is top push the button on the microphone. What the button does is pull the PTT relay in the radio to ground, thus setting it into transmit mode. For the case of this system what will be done is that through one of the GPIO pins of the Raspberry Pi 3 and a PTT circuit in the interface board, the MCU will ground the relay and set the radio into transmit mode.

Since the IC-2A VHF Aircraft radio is a half-duplex communication system it can only do one of the two communication functions at a time. When the PTT is not grounded the KX 170B is in 'Receive Mode" and can receive incoming audio signals. But when the PTT is grounded the radio switches to 'Transmit Mode'. In this mode, the system cannot process any received audio and any communication to it is essentially lost.

3.2.2.5 Automatic Gain Control (AGC)

Automatic Gain Control is a closed loop-feedback circuit where a signal is fed into and it's expected to maintain and regulate to certain level of amplification. This signal can be sound or radio frequency. The AGC can give us two different cases for output. The first case is if the level of the input signal is too low, the designed system will output an amplified signal to the desired level. The second case is if the input signal is too high, the designed system will output a lowered signal to the desired level as well. The purpose of this system is to maintain a constant level for the output signal giving a wider range of input signal levels. AGC is commonly used is radio receiving to help equalize the desired average volume due to different levels received in the strength of signals and fading of the same. One of the consequences of not using an AGC is seen in the relationship between the signal amplitude and the sound waveform – the amplitude of this signal is proportional to the radio signal amplitude. The information contained by the signal is carried by the changes of the amplitude of the carrier wave. If the circuit were not linear, the modulated signal could not be recovered with reasonable fidelity. However, the strength of the signal received will vary widely, depending on the power and distance of the transmitter, and signal path attenuation. Overall, the AGC circuit keeps the receiver's output level from fluctuating too much by detecting the overall strength of the signal and automatically adjusting the gain of the receiver to maintain the output level within an acceptable range.

3.2.3 Language Options and Selection

The MCU chosen also impacted our choice of programming language. This section describes the language chosen, why it was chosen, and how it will impact the system.

3.2.3.1 MCU

For the main control unit, or MCU, there are a few options as far as what language to choose. Since we are utilizing the Raspberry Pi 3 for the MCU the first priority, was making sure that the programming language that we selected was directly compatible with the Raspberry Pi and had libraries in which to access the multiple General Purpose Input and Output pins, or GPIO pins. Having a library for the Raspberry Pi's GPIO pins allows us to not have to work from the ground up, and strictly focus on how we are going to program the GPIO pins specifically. This saves us a lot of time and effort that we don't have to put into a lot of code that's only purpose would be to allow us to access the pins.

For this design, we chose to go with Python as our programming language for the MCU. Using python solves the initial requirement of having a default library for interfacing with the Raspberry Pi's GPIO pins through the RPI.GPIO library. This allows for basic reading

and writing to the pins without having to create those initial functions ourselves.

Another reason we chose python as our main language was because the Analog to Digital Converter, known as an ADC, that we are choosing has a python library that allow for easier reading and writing directly from the chip without having to do a lot of initial handshake messages and procedures to receive and send data over I2C. Because reading from most particular ADC's can be complicated, as they have certain bit patterns in which are needed to configure and choose which of the devices' functions are being used, it is nice to have an extra bit of encapsulation in which instead of building these bit patterns ourselves, we can simple call a read or write method. This not only shortens the amount of code written by us but again allows us to focus more on the actual implementation of our system rather than having to deal with a lot of headache simply reading from the ADC. This library is also open source so it is free to use and heavily supported by the community in case we run into any issues.

Python was a good choice compared to other languages such as C as not only is it inherently Object-Oriented and allows for a more modular structure to our code. The Object-Oriented nature of Python allows us to create objects in which to delegate the functions of reading and writing to certain components and sensors. It will also allow us to create a "Weather" object to collect the current conditions to easily pass them to the main function which will create the audio to transmit to the pilot. This will simplify the code immensely and make it simple to add new weather reading as necessary. The Object-Oriented nature of Python also allows us to give control of certain components to certain objects and much more easily debug our code. Python is also a scripting language which makes it highly flexible in where and how it is implemented. This means that no matter how we structure the system and integrate the various other components (i.e. the HTTP server, DCHP server, etc.) the usage of our code can be kept relatively independent. This allows us more freedom to change certain modules and components in the system if we must and not have to overhaul our python scripts too much. In other languages like C, it can be much more difficult to configure the code with these different components, as it has to be recompiled and is only set to run a certain way. There is not a lot of flexibility there, which is ideally what we find to be valuable in the structure of this system.

3.2.4 Text-to-Voice Software

One of the most significant component to our system design is the Text to Speech software. This software style, abbreviated as TTS, is a form of speech synthesis created use a variety of text to fully automate and convert those text into spoken voice output. It

basically obtains all the weather and transmission data obtained from all our components sensors and creates a voiced broadcast that will be used to communicate that information. The user may simply also request a communications check which then does not make use of the TTS but instead creates a playback which records and rebroadcasts the previously transmitted information providing a power level to that transmission as well. Both broadcasts are played over the radio channel and heard by the pilot after punching in the correct click pattern. In order to produce a clear and coherently pronounce the provided key words a few important requirements had to be met when selecting the correct text-to-speech software. The main priority is that the speech software we utilize would have to always keep providing the pilot with bullet clear and concise data at all times. The clearness must persist even when the speech modules is creates using the simple audible outputs over normal laptop speakers. This speakers' signal usually run at different amplification and compression circuits which are then eventually finally broadcasted of the radio channel as radio waves.

Furthermore, during the process of processing the sensors data and recording and recommunicating the communications check data meant to be replayed to the pilot, our system is consistently synthesizing speech by concatenating sentences from a self-provided database of prerecorded words. The voice response system is limited to the response it can provide base on this database of words predetermined for the system. In addition, throughout this process the system maybe heavily infected by a lot of interferences and might become disoriented before it is heard by the pilot failing the requirement of providing a clear and concise voiced-over message (with no noise) to any inputs selected by the pilot. And thus, it is really important to clear and clean the output as it suffers from many possible interferences. Another major important requirement for this speech software is that it provides a not too fast verbal response to a provided input as so to not mispronounce or cause the pilot to miss hear the information if the software answers in a faster tempo. We needed to find the correct voice that would response sophistically enough and articulate every word encountered.

3.2.4.1 Requirements for The Text-to-Speech Software

For our project, we also wanted to offer a language software that would provide multiple languages and allow the user to adjust different settings. These different settings would encompass allowing the user to program multiple languages, pronunciation, and

also allow for customizing the speed of the output signal. As an output is non-acceptable if it is broadcasted too fast to hear or mispronounces certain words. In addition, we needed to research different software applications and allow our sponsor to listen and hand-picked the voice pronunciation that would best meet his pronunciation and aptitude requirement. The voice settings most also be appealing enough to most other users' intent on using this system in order to improves the user's experience creating an ease of use with the system. The next requirement on the list is for the Text to Speech software to have the ability to easily save and store the output in a file. This can be utilized to test the system and log a history of all the inputs up to a certain point. This way, the system keeps track of a list of requested inputs and outputs in case the user wants to observe previous broadcasts.

One last requirement, probably one of the most important, is that whichever of the multiple open sourced Text to Speech solutions we select must be accessible even without internet connection. If the system is placed within an area where a solid internet connection is unreliable it should still be able to output the voiced over information requested. In that case, we decided not to have a major part of the system be reliant on something as a strong internet connection in order to function properly. It is best if the software installed does not demand internet connection in order to service the user. Using the listed requirements above, we ran across a few good Text to Speech solutions. Among them is IVONA Text, this text to speech solution that supports both SSML 1.0 and 1.1 (as defined by speech synthesis markup languages standards). IVONA text provided by far the clearest and best voice out of all the other Text to Speech software we came across. It provided great functionalities and was highly configurable providing many configurations that allowed its user to set the nationality, language, gender, and even pronunciation method. At first, we we're very ecstatic that we found such a system that provided so much customizability and we completely overlooked one of the requirements (actually describe as a major requirement) listed above. We needed a software system that would not require a reliable internet connection to function properly. Another apparent and incredibly further annoying issue that moved us away from this software is the other fact that it breaks yet again another requirement by not providing a possible way of easily saving the output of the file by default. Even worst when we realized we we're

looking at a software system that required a monthly payment service. We then added the requirement that the system must be free as our sponsor would definitely not wish to pay a periodic sum per month for this software.

3.2.4.2 Selecting the Text-to-Speech Software

Looking further for a free test to speech software, we came across the Festival TTS. We made sure that this was a possibility by first simply asking if it was free, open sourced, and mainly also compatible with a Linux system. The Festival TTS software is written in using C++ libraries and provide a general framework for building speech synthesis systems. It also includes various modules that offer full text to speech from a number of APIs. Festival TTS as of this moment is only bilingual providing an interpreter for English and Spanish. This is purposely fine for our case as we only require a system that can work in English. Festival works well on Linus and is by far the most configurable system we found as it provided us with tons of different configurable voices. Furthermore, the online community created and uploaded a multitude of other language packs that can simply be imported into the system that are neatly documented. The harsh compatibility issue to one requirement that needs to be met to pair well with our system was that Festival was not as clear as we wanted nor provided an easily storable filesystem. Another set-back that causes keep researching and testing different text to speech software.

Finally, we came across the PICO TTS and hoped it would meet all of our specified requirements. The PICO TTS is a barebones and stripped down version of an abandoned text to speech project recently used in googles android products that was formerly named Google TTS. This software provided incomparable voice quality with a lot of support and documentation. The Google TTS was scrapped and switched into PICO which is a free open source, non-commercial product that boast of being an improvement over Festival, PICO, and FLITE (another TTS). PICO is also open source (just like GOOGLE TTS used to be) and run quite perfectly on Linux and the Raspbian operating system of the Raspberry Pi. With Linux, the installation step is quite simple as we only need to call the commands using a terminal which facilitates the installation process by making it overly easy. While there is not a ton of configuration for this system, it doesn't require internet

connection, is open source, and above all free. Furthermore, we finally settled on this choice because it also fulfilled all our other requirements. It provided a clear voice output and the file is easily store as a wav file. This system lacks the configurability of the other TTS's mentioned above but at least still provides a way configure the actual voice over. The default gender which is set to a female voice and cannot be changed. This is also fine as our sponsor declared that he would prefer to have a female voice with a sort of clear accent. Thus, this is not an issue for our project it fits perfectly within the scope of what we wish to accomplish. It's true that the PICO doesn't provide much configurability in the voice department, but at least provides a good amount of different languages while also allowing the user to switch the pronunciation speed with different filters. This can be changed by editing the text that is being sent to the engine. The PICO TTS engine provides us with just enough configurability, it is free, and runs quite well on the Linux operating system without needing an internet connection. This system evidently meets all our requirements and was thus the clear winner for our project.

3.2.3 Voltage Regulator Options and Selections

The power supply system of the Auto FBO needs three supply voltages of 3.3, 5, and 13.7 or 15 V. The 3.3 V supply will need to supply an estimated maximum current of 0.54 A, the 5 V supply 2.83 A, and the 15 V supply 1.44 A. All regulators under $10.00 were considered to aid in the overall price of the Auto FBO system. The tables 3.2.3.1 – 3.2.3.3 below show a comparison of the final selection of regulators considered, with the chosen regulators marked with a star after their part number.

The main parameters chosen to compare the candidate linear voltage regulators for the regular system were the regulated voltage range, maximum current output, maximum input voltage, maximum voltage dropout at the maximum current output, and price. The regulated voltage range is the given voltage range that a regulator will have at its output, at or near the maximum output current. The regulated voltage range is an important parameter to consider because a wide regulated voltage range can cause unstable or unforeseen effects on other components it is supplying, which usually have a minimum and maximum supply voltage specification. Maximum current output was considered since the power supply system must be dependable enough to deliver enough current to all devices if they are demanding maximum current. The maximum input voltage and

maximum dropout voltage go hand-in-hand. The dropout voltage of a voltage regulator is the smallest possible difference between the input voltage and output voltage for the regulator to remain in its intended operating range. For example, a regulator with a 15 V output and a 2 V dropout voltage rating will only output 15 V if the input voltage is above 17 V. If the input falls below 17 V the output of fail to regulate 15 V. The maximum input voltage is important for all regulators because the 15 V regulators of this design will have around a 2 V dropout voltage. Due to this, the maximum input voltage of any regulator to be considered must be around 17 volts, however an input voltage greater than 17 volts would be preferred to provide overhead. As demonstrated in the example above the lower the maximum dropout voltage the more dependable a regulator it will be. Other parameters such as the line regulation, load regulation, maximum quiescent current, and operating temperature are used as well to decide which linear regulator to choose. However, these parameters carried less weight than the formerly described parameters, and we're only included for a more well-rounded comparison.

In choosing the 3 V linear regulator it was an obvious choice to choose the LT1129I-3.3 since its maximum input voltage is 30 volts and the other two regulators had only a 16 V maximum input voltage. This regulator also met the maximum current output needed for the regulator design. These qualifications along with its other specifications and price gave merit to choose this regulator. The decision between picking a 15 V or 13.7 V regulator was made when searching for a 13.7 V regulator. The only 13.7 regulator found commercially available had suitable specifications, however, not many parts for left on the market. The 15 V regulator was chosen for reliability of buying instead. Since each of the candidate 15 V regulators had a maximum input voltage of 35 V, the L7815C regulator was chosen since it had a lower maximum voltage drop out with enough maximum output current with a tighter regulated voltage.

The main parameters chosen to compare the candidate switching voltage regulators were the efficiency, maximum current output, maximum input voltage, voltage regulation, switching frequency, switch resistance, and maximum Q current. Efficiency is highly important to conserve power as the input voltage of the device will be 20V with a 5V output voltage. Maximum current output was considered since the power supply system must be dependable enough to deliver enough current to all devices if they are demanding maximum current. Maximum input voltage was considered since the input voltage to the regulator is set to be 20V. The voltage regulation is highly important as this regulator is primarily supplying the RPI. The switching frequency and resistance were taken into consideration since a higher switching frequency along with a low resistance would create a tighter DC voltage rail with minimum voltage drop. The LM2676 was chosen because

of its maximum output current and price compared to the LM2670.

3.2.3 AC to DC Power Supply Options and Selections

The candidates for the central power supply were chosen to supply at least a maximum current output of 5 A, the maximum demand of the design, and a supply voltage of 20 V, as required for the voltage regulator system. This voltage and current was chosen to prevent against dropout of the 15V linear voltage regulator and provide maximum current demands of the design. Also, the power supply units were only chosen if their price for one unit was under $60.00 Shown in table 3.2.3.4 are the power supply units considered for the central power supply with their specifications as well as price, with the chosen unit marked with a star after its part number.

The main parameters chosen to compare power supply to units where AC input voltage, DC output voltage, maximum current output, efficiency, and price. Since the 15V linear regulator was going to have the most power dissipation with a dropout voltage of 2V, a supply voltage of 17–20 V was needed. This constraint narrowed the search or a power supply unit vastly, especially considering price. The two considerations for the power supply unit we're from the same company with similar design. The GST120A20-R7B power supply unit was chosen since it had the cheapest price and the necessary specifications.

Part No.	Min-Max Regulated Voltage	Max Current Output	Max Input Voltage	Max Voltage Dropout at Max Current Output	Line Regulation	Max Load Regulation	Max Q Current	Operating Temp.	Per Unit Price
	V	A	V	-	-	-	mA	°C	
TLV1117I-33	3.168-3.432	0.8	16	1.2	10 mV (max)	15 mV (Max)	15	-40-125	$0.85
LT1129I-3.3*	3.250-3.350	0.7	30	0.7	10 mV (max)	30 mV (Max)	.050	-40-125	$5.65
AMD7150	±2%	0.8	16	1	±0.01 %	1%	4.3	-40-125	$4.91

Table 3.2.3.1: 3.3 V Linear Regulator Comparison

Part No.	Max Efficiency	Max Current Output	Max Input Voltage	Min-Max Regulated Voltage	Frequency	Switch Resistance	Max Q Current	Operating Temp.	Per Unit Price
	%	A	V	V	kHz	Ω	mA	°C	
LM2676*	94	3	45	4.9 - 5.1	260	0.15	6	-40 -125	$4.90
LM2670	94	3	40	4.9 - 5.1	260	0.15	6	-40 -125	$6.00
LM53625	90	2.5	36	4.92 – 5.125	2100	0.13	0.16	-40 - 125	$3.70

Table 3.2.3.2: 5 V Switching Regulator Comparison

Part No.	Min-Max Regulated Voltage (V)	Max Current Output (A)	Max Input Voltage (V)	Max Voltage Dropout at Max Current Output (V)	Line Regulation (mV)	Max Load Regulation (mV)	Max Q Current (mA)	Operating Temperature (°C)	Per Unit Price
L78S15C	14.25-15.75	2	35	2.5	300	150	8	0-150	$0.84
L7815C*	14.4-15.6	1.5	35	2	150	100	6	-40-125	$0.61
LM340	14.25-15.75	1.5	35	2	150	150	8.5	0-125	$1.51

Table 3.2.3.3: 15 V Linear Regulator Comparison

Power Supply Unit	Input Voltage (VAC)	Output Voltage (VDC)	Max Output Current (A)	Efficiency	Overload Protection	Overvoltage Protection	Output Power (W)	Operating Temperature (°C)	Per Unit Price
GSM160B20-R7B	80-264	20	8	92.5%	105-150%	105-135%	160	-30-70	$61.75
GST120A20-R7B*	85-264	20	6	90%	105-160%	105-135%	120	-30-70	$41.68

Table 3.2.3.4: AC/DC Central Power Supply Unit Comparison

3.2.4 Weather Sensor Options and Selections

In order to meet the specifications for the weather system it is necessary to select devices that can measure wind direction and speed, temperature, dew point, and pressure. The sensing of wind speed and direction is typically measured by an anemometer and wind vane. There several types of these devices including cup, vane, hot-wire, laser doppler, and ultrasonic anemometers. Temperature is measured by a thermometer which is also used for the measurement of dew point which utilizes humidity and temperature. For our weather reporting system, pressure will need to be reported as absolute pressure. Current pressure sensing technology includes vizio resistive strain gauge, capacitive, electromagnetic, and potentiometric. For the purposes of this design it was desirable to choose weather sensors that would communicate in I2C.

3.2.4.1 Anemometer and Wind Vane

The anemometer and wind vane huge for the weather system is the Davis Instruments 7911 Anemometer. This device is used as it was given to this project free of charge by our adviser. The 7911 Anemometer features 3 polycarbonate wind cups to measure wind speed and a UV-resistant ABS plastic wind vane to measure wind direction. It comes with a 40- foot long, 26 AWG cable that ends with an RJ-11 connector. It can measure wind speeds up to 173 knots (200 mph) with a 1 knot resolution and a ±5% accuracy. It can also measure wind direction from 0 degrees to 360 degrees with a 1-degree resolution and a ±7% accuracy. The Davis Instruments 7911 Anemometer is also a component of the Weather Monitor II and Weather Wizard III, both of which are complete weather stations also manufactured by Davis Instruments.

3.2.4.2 THD Sensors

A comparison among the potential temperature, humidity, and dew point sensors are shown in the tables below. Since dew point can be derived from temperature and humidity measurements only temperature and humidity sensors are necessary for the weather system. The main parameters used for comparison among the temperature sensors are range, accuracy, resolution, long term stability, maximum response time, voltage supply, maximum current use, operating temperature, and price. Similarly, the main parameters used for humidity sensors mirror that of the temperature sensors. The chosen THD (Temperature Humidity Dew Point) sensor is marked with a star in the tables below after its part number.

3.2.4.3 Barometric Sensor

A comparison among the potential barometers are shown in tables below with the chosen sensor marked with a star in the table below after its part number. Barometers were only chosen if they only had a range of roughly 20-40 inHg, as this is slightly beyond the range of atmospheric pressure around the world. The main parameters used for comparison among the barometers are similar to that of the temperature and humidity sensors.

3.2.4.4 MS860702BA01

Among all the temperature, humidity, and pressure sensors the chosen device to cover these measurements was the MS860702BA1. Not only was it chosen because it could be used for temperature, humidity, and pressure measurements, but also its specifications compared to the other parts. In terms of price, however, it is clearly a better selection, especially if mass production of this system is to be implemented.

The MS8607 is the novel digital combination sensor of MEAS providing 3 environmental physical measurements all-in-one: pressure, humidity and temperature (PHT). This product is optimal for applications in which key requirements such as ultra low power consumption, high PHT accuracy and compactness are critical. High pressure resolution combined with high PHT linearity makes the MS8607 an ideal candidate for environmental monitoring and altimeter in smart phones and tablet PC, as well as PHT applications such as HVAC and weather stations. This new sensor module generation is based on leading MEMS technologies and latest benefits from Measurement Specialties proven experience and know-how in high volume manufacturing of sensor modules, which has been widely used for over a decade.

Regarding its temperature measurements, the MS860702BA1 performs best among the other parts in max response time and power consumption. Its temperature range is third best, however, its range is more than adequate. The accuracy of the device is the worst among the selected devices, but is sufficient enough for accurate weather reporting. Resolution is among the best, along with its long-term stability. The humidity and pressure specifications of the device is overall the best out of all the possible selections.

The MS8607 includes two sensors with distinctive MEMS technologies to measure pressure, humidity and temperature. The first sensor is a piezo-resistive sensor providing pressure and temperature. The second sensor is a capacitive type humidity sensor providing relative humidity. Each sensor is interfaced to a ΔΣ ADC integrated

circuit for the digital conversion. The MS8607 converts both analog output voltages to a 24-bit digital value for the pressure and temperature measurements, and a 12-bit digital value for the relative humidity measurement.

Another reason this sensor was selected was because it can be communicated with via I2C. Since the anemometer uses the same communication protocol, it greatly simplifies integration if both sensors run on the same protocol. The external microcontroller clocks in the data through the input SCL (Serial CLock) and SDA (Serial DAta). Both sensors respond on the same pin SDA which is bidirectional for the I2C bus interface. Two distinct I2C addresses are used (one for pressure and temperature, the other for relative humidity). The I2C address for pressure and temperature is 1110110, while the I2C address for humidity is 1000000.

Part No.	Range (°C)	Accuracy (°C)	Resolution (°C)	Long Term Stability (°C/year)	Max Response Period (s)	Voltage Supply (V)	Max Current Use (mA)	Operating Temperature (°C)	Per Unit Price
DHT22	-40-80	±0.5	0.1	N/A	2	3.3-6	2.5	-40-80	$9.95
HDC1080	-40-125	±0.2	0.1	N/A	0.0064	2.7-5.5	7.2	-40-125	$4.65
SHT21	-40-125	±0.3	0.01	< 0.02	5-30	2.1-3.6	0.330	-40-125	$6.62
MS8607-02BA01*	-40-85	±1	0.01	±0.3	0.015	1.5-3.6	1.25	-40-85	$8.48

Table 3.2.4.1: Temperature Sensor Comparison

Part No.	Range	Accuracy	Resolution	Stability (RH% /year)	Max Response Period (s)	Voltage Supply (V)	Max Current Use (mA)	Operating Temperature (°C)	Per Unit Price
DHT22	0-100%	2-5%	0.1%	±0.5%	2	3.3-6	2.5	-40-80	$9.95
HDC1080	0-100%	±2%	0.1%	±0.25%	0.0065	2.7-5.5	7.2	-20-70	$4.65
SHT21	0-100%	±2%	0.04%	<0.25%	8	2.1-3.6	0.330	-40-125	$6.62
MS8607-02BA01*	0-100%	±3%	0.04%	±0.5%	0.015	1.5-3.6	1.25	-40-85	$8.48

Table 3.2.4.2: Humidity Sensor Comparison

Part No.	Range (inHg)	Accuracy (inHg)	Resolution (inHg)	Long Term Stability (inHg/year)	Max Response Period (s)	Voltage Supply (V)	Max Current (mA)	Operating Temp (°C)	Per Unit Price
KP236N6165	17.718-48.7245	±0.2953	0.2953	N/A	0.010	4.5-5.5	10	-40-125	$6.80
MPL3155A2	14.765-32.483	±0.4	0.00044	±0.295	0.512	3-5.5	2	-40-85	$9.95

MS8607-02BA01*	0.2953-59.06	±0.059	0.0005	±0.0295	0.015	1.5-3.6	1.25	-40-85	$8.48

Table 3.2.4.3: Pressure Sensor Comparison

3.2.5 Operational Amplifier Options and Selections

The usage of operational amplifiers in the Auto FBO system will be exclusively for audio signals. These signals have a bandwidth of roughly 0 to 20 kHz. A quality operational amplifier will have the basic requirements of low noise, low total harmonic distortion (THD), good response (slew rate), and low power. However, these are somewhat conflicting requirements. Typically, lower power operational amplifiers with have poor noise and THD specifications.

Table 3.2.5 compares the potential operational amplifiers used for the Auto FBO system. For this comparison noise, slew rate, gain bandwidth product, total harmonic distortion, supply voltage and current, CMRR, and price per unit are included. The main factors in this comparison are noise, THD, and slew rate. During the recording and playback of the voice communication check, our system strives to not change the incoming audio signal in any way. Thus, low noise and THD is needed along with a good response rate. The GDP, CMRR, and supply voltage and current are also important features to the characterization of an operational amplifier, and were thus included. Cost is also of concern as our goal is to produce a low-cost product. However, a higher performance device will obviously cost a lot more.

The chosen operational amplifier was the NE5534A. It was determined that the needed slew rate for audio signals up to 20 kHz was 0.377 μs/V. This was determined by the equation $SR = 2\pi fV$ where f is the maximum frequency of interest and V is the max voltage. This slew rate was met by all the chosen candidate operational amplifiers, but some overhead was preferable. The NE5524A also has a great noise figure even comparable with the high performance OPA models. These factors along with its other specifications and low price is why this operational amplifier was chosen.

Part Number	Noise	Slew Rate	GBP	THD+N	Supply Voltage	Supply Current	CMRR	Price Per Unit
	nV/√Hz (1kHz)	V/µs	MHz	%	V	mA	dB	
TL082	18	13	3	0.003	7-32	2.2	100	0.50
OPA2314	14	1.5	3	0.001	1.8-5.5	0.15	96	0.75
OPA2376	7.5	2	5.5	0.00027	2.2-5.5	0.76	90	1.20
NE5534A*	3.5	13	10	0.002	6-40	4	100	0.90
OPA209	2.2	6.4	18	0.000025	4.5-36	2.5	130	1.50
OPA1612	1.1	27	27	0.000015	4.5-36	3.6	120	5.00
LM4562	2.7	20	55	0.00003	5-34	4.8	120	3.00
LME49726	15	3.7	6.25	0.00008	2.5-5.5	0.18	98	0.80
NJM2060	10	4	10	0.01	8-36	2.25	90	0.43
LM833	4.5	7	16	0.002	10-36	2.05	100	0.40

Table 3.2.5: Operational Amplifier Comparison

3.3 VHF Aircraft Radio Selection

The ICOM IC-A2 is a compact, synthesizes, 5 W PEP, VHF handheld transceiver. The IC-A2 offers keyboard frequency selection with extremely good stability and frequency accuracy. Shown in figure 3.5 below is the ICOM IC-A2.

Figure 3.5 ICOM IC-A2

3.4 Termination of Unused Operational Amplifiers

When using a dual or quad operational amplifier device it is common to have an extra operational amplifier stage left over that isn't required by other circuits in the design. In this case, it is critical to correctly terminate the device. By terminate, we mean to configure the device in a manner that allows for it to operate in a stable and predictable manner. The added benefits of proper termination are reduced susceptibility to noise, reduced input power consumption, reduced power dissipation, and reduced exposure to EOS.

The understanding of an operational amplifiers specifications will aid in properly terminating a device. These specifications include input common-mode voltage range and input differential voltage range. The input common-mode voltage range is the input rage for which a stable linear behavior is guaranteed. The input differential voltage range is the max voltage allowed between input pins. Exceeding this range can overstress the input stage. Concerning the output stage of the amplifier, the output stage can saturate when driven to either supply rail. When saturated to operational amplifier will consume more power than if it was not saturated. Since operational amplifiers have large open-loop gain, negative feedback is recommended to achieve a low, stable, and predictable behavior.

Shown below in figures 3.4a and 3.4b are the proper configurations to terminate unused operational amplifiers. The overall goal is to keep the output voltage directly between the positive and negative supply rails. Both configurations make use of a voltage follower topography.

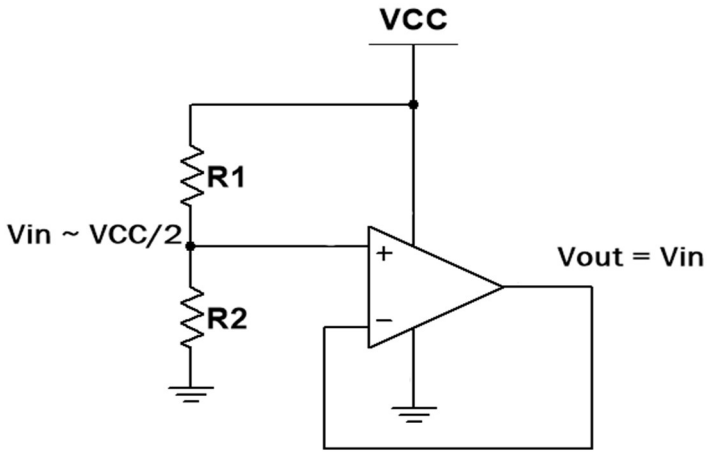


Figure 3.4a: Single Supply Termination

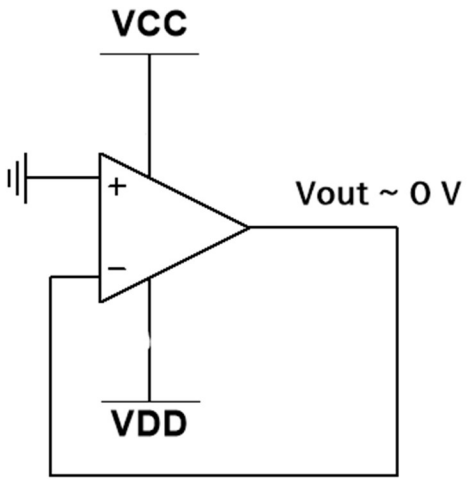


Figure 3.4b: Dual Supply Termination

3.5 Circuit Protection

Transient Voltage Suppressors, TVS, are devices used to protect vulnerable circuits from electrical overstress such as that caused by electrostatic discharge, inductive load switching and induced lightning. Within the TVS, damaging voltage spikes are limited by clamping or avalanche action of a rugged silicon pn junction which reduces the amplitude of the transient to a nondestructive level. In a circuit, the TVS should be invisible until a transient appears. Electrical parameters such as breakdown voltage(VBR), standby (leakage) current (ID), and capacitance should have no effect on normal circuit performance. When used in circuit design TVS are put in parallel with loads as shown in figure 3.5.

One scenario where TVS can help protect electrical devices is lightning strikes. Even though a direct strike is clearly destructive, transients induced by lightning are not the result of a direct strike. When a lightning strike occurs, the event creates a magnetic field which can induce transients of large magnitude in nearby electrical cables. A cloud-to-cloud strike will affect not only overhead cables, but also buried cables. Even a strike 1 mile distant (1.6km) can generate 70 volts in electrical cables.

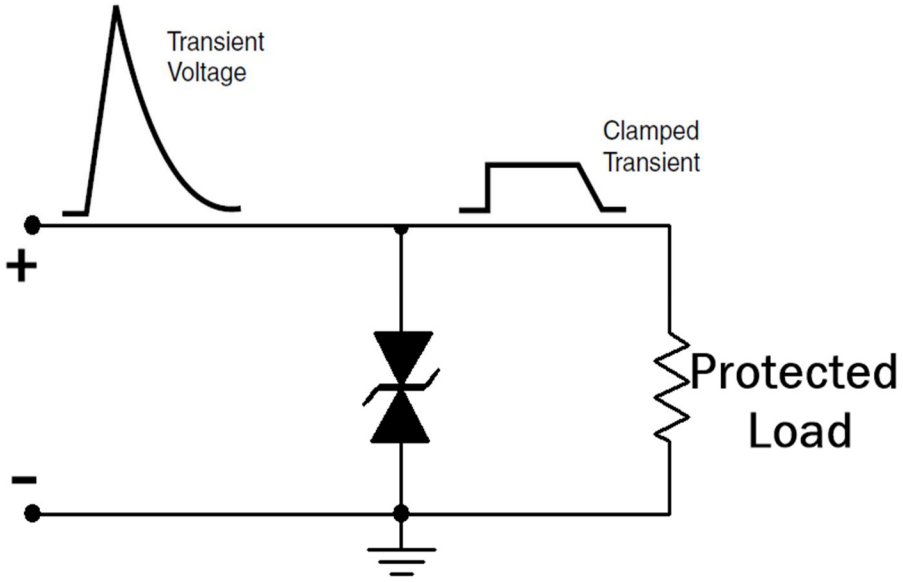


Figure 3.5: TVS Application

3.6 Interfaces

This section details how the various components of the system will communicate with each other. Given the nature of the system, there are many streams of data that need to be accurately relayed from one component to another so the interfaces between them are crucial.

3.6.1 To Radio

These are the signals the radio will transmit to the pilots so the audio coming to the radio needs to be in a form that it can transmit and it cannot be distorted.

3.6.1.1 TX Audio

The transmission will be an analog audio output coming from the audio CODEC we will implement on the interface board. This audio will be sent through a low pass filter to remove any high frequency noise added from the raspberry pi before being sent to the radio through voltage-follower circuit to remove any loading effect.

3.6.1.2 PTT

This PTT block will put the radio into transmit mode prior to audio being sent to it. The purpose of this signal is to simulate the action that is pushing the mic button to talk over the radio.

3.6.2 From Radio

Like the signals being used to be able to transmit through the IC-2A Radio, there is also a need to analyze the signals coming from it. These signals will allow for the Raspberry Pi 3 to analyze what is needed by the pilot at the other end, as well as allow the Raspberry Pi to receive the actual audio from the pilot.

3.6.2.1 RX Audio

As previously mentioned the Raspberry Pi will need to be able to receive the audio being transmitted by the pilot. This RX audio signal will be picked up from the top of the volume potentiometer and run through the interface board. This is to prevent the volume setting on the actual radio to affect the RX audio signal being transmitted to the interface board. Once the audio signal is received on the interface board it will be sent to a unity gain buffer and then sent to our codec chip which will amplify and digitize the signal into a Pulse Code Modulated signal which can then be sent to our raspberry pi for recording.

3.6.2.2 Carrier Detect

The carrier detect in our system was identified from the main radio. The carrier detect levels were obtained by connecting the radio, at the squelch circuit output, to the oscilloscope and examining the output voltage when there is a radio signal detect present and when there is no radio signal detect present. For our radio, we found that when there is no carrier present our squelch voltage is 0 V, and when there is a strong enough signal detected the squelch voltage jumps to 4.8 V.

3.6.3 From Microcomputer

The only two signals coming from our raspberry pi will be the PTT and TX audio signals. These will be received by our IC-2A radio and utilized to broadcast back to the user on the other end of the communication channel.

3.6.3.1 PTT

The start of the PTT line will be originated from one of the Raspberry Pi's GPIO pins which will be fed to our interface board which can then be pulled to ground to signal the radio to begin transmission.

3.6.3.2 TX Audio

The audio will come out from the Raspberry Pi through Pulse Code Modulated lines that will then be sent to the audio codec for decoding and transforming into an analog signal that will be useful for the radio to receive.

3.6.4 To Microcomputer

These are the signals sent from the radio and weather sensors to the microcontroller for processing. The weather sensors need to be easily accessible and the carrier signals need to be real time and undistorted.

3.6.4.1 I2C Bus

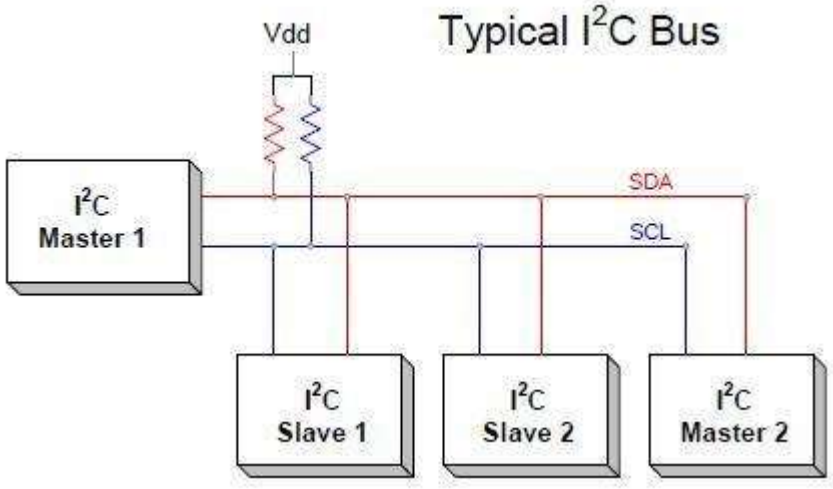


Figure 3.3.4 Typical I2C Configuration

All communication with peripheral devices will be interfaced over the I2C "I-squared-C" bus that is able to individually address each device. A typical configuration is shown in Figure 3.3.4. Currently the ADC (handling the wind speed/direction and AGC), the audio codec communication, and the temperature/humidity/pressure sensor will use the I2C bus to communicate with the Raspberry Pi. Currently, the Raspberry Pi will act as the Master providing the clock for all devices configured to be slaves.

3.6.4.2 Carrier Detect

The squelch voltage will be handled on our interface board by using a comparator to check and see if the voltage has risen above a set value. In this case our squelch, when on, goes to 4.8 V which we will compare to a 3 V baseline. When the squelch turns on the comparator will send a logical output of 1 to the Raspberry Pi where it can be distinguished from the "off" reading of 0 volts.

3.6.5 From Anemometer

From the anemometer, we will be sending two signals one for the wind speed and one for the wind direction. For the wind direction, we simply supply the anemometer with a 3 volt signal and the anemometer uses an internal potentiometer to range the voltage from 3 V – 0 V. Next, we send the wind speed line directly into the Raspberry Pi where it will

pulse low to indicate one rotation. The microcontroller will the determine how many clicks occur in a given timeframe to determine the wind speed measurement in knots.

3.7 Carrier Detect

The consolidation between the Radio and Interface Board serves as the bridge to be able to condition the carrier detect and identify when there will be transmission. Since we only have two (2) levels for identification, a comparator is being used to compare and determine which level, that indicates transmission or no transmission, is being received.

The comparator being used is the LM393 Dual Differential Comparator. The purpose of this device is to compare two (2) voltage values, and output a digital signal indicating which of the two is larger to the main control unit through a GPIO.

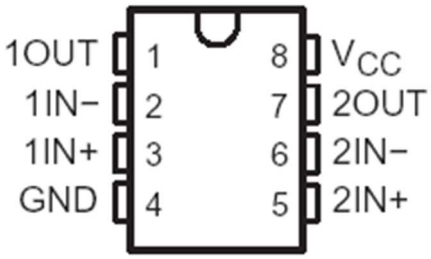


Figure 4.2.2 LM393

The differential comparator consists of a high gain differential amplifier. These devices are commonly used in systems that measure and digitize analog signals such as analog to digital converters, as well as relaxation oscillators. In our application, we compare the received signal, carrier detect present or carrier detect not present, with a reference voltage.

3.7.1 Automatic Gain Control Voltage

The AGC Voltage will be fed to our ADC where it will then be turned into a digital signal useful by the Raspberry Pi to give Power Level Received readings back to the pilot. Included in Figure 4.2.5.1 are the correlations we made between input signal strength and AGC Voltage Levels. This will be used by the software as a lookup table to determine what reading to give back to the pilot. The range of inputs (measured) for the amplitude gain control from the radio are the following:

Signal Power (dBm)	AGC Voltage (V)
-120	3.43
-117	3.43
-114	3.43
-111	3.36
-108	3.115
-105	2.94
-102	2.745
-99	2.455
-96	2.315
-93	2.19
-90	2.045
-87	1.93
-84	1.84
-81	1.75
-78	1.66
-75	1.62

Signal Power (dBm)	AGC Voltage (V)
-72	1.59
-69	1.56
-66	1.54
-63	1.52
-60	1.49
-57	1.47
-54	1.45
-51	1.43
-48	1.40
-45	1.38
-42	1.36
-39	1.33
-36	1.32
-33	1.3
-30	1.28

Figure 4.2.5.1

4. Design Constraints and Standards

This chapter will define all the standards and any design constraints that apply to the Auto FBO system.

4.1 Standards

This section describes relevant standards that apply to the Auto FBO system. Each of these standards were used in order to keep the system as easy to set up and compatible as possible. It would not have been efficient to design a system with standards that are not well known or well supported.

4.1.1 Registered Jack Standard

A Registered Jack (RJ) is a standardized network interface for connecting data and signal equipment, usually over a long distance. The RJ is defined in the international standard for physical network interfaces. This standard includes specifications of physical construction, writing, and signal semantics. The interfaces defined in the RJ standard include RJ-11, RJ-14, RJ-21, RJ-45, and the RJ-48 connector types, as well as many other types.

The most current version of the standard is TIA-968-A. This specification defines the modular connection fully, but not the wiring. The wiring specification is instead included in the standard T1.TR5-1999, "Network and Customer Installation Interface Connector Wiring Configuration Catalog". With the addition of the publication of the TIA-968-B standard, the connector specification has been moved to TIA-968-A.

Each registered jack type, such as RJ11, identifies both the physical connectors and the wiring. Thus, an inspection of the connector type will not necessarily indicate the type of wiring used in the cable. This is because the same connector can be used for a multitude of wiring patterns. This has led many confusion among the industry and its customers of what type of cable standard is actually being used in an application. For example, the RJ11 connector is also used for the RJ14. Tale 4.1.1 below shows a few of the officially recognized registered jacks with their connectors. Most registered jacks use designation XPYC, where X is the number of positions on the connector and Y denotes the number of conductors. For example, the RJ11 can use a 6P4C connector where there

are 6 positions and 4 conductor connections. The RJ11 6P4C connector is shown in Figure 4.1.1a.

Code	Connector	Note
RJ11	6P2C	Common usage in single telephone lines, 6P4C can also be used
RJ21X	50-pin micro ribbon	Up to 25 lines
RJ45S	8P8C keyed	One data line with programming resistor
RJ48C	8P4C	Four-wire data line

Table 4.1.1

Figure 4.1.1a

Typical wiring of registered jacks uses twisted pairs with separation of supply and data lines with ground lines. These conventions were originally put in place to help create a standard of wiring across the industry. The pinouts of the connectors of each registered jack usually correlate to a specific function for a given application and are color coordinated as shown in Figure 4.1.1b.

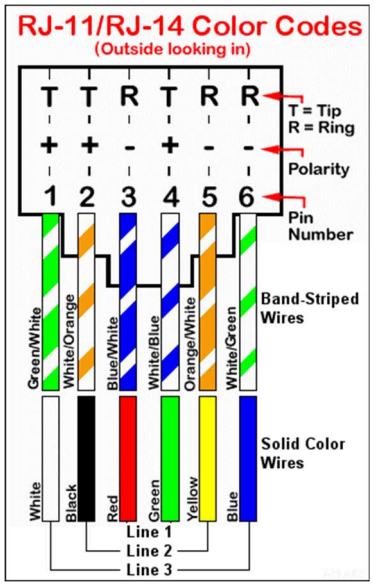


Figure 4.1.1b

4.1.2 Radio Communication Phraseology and Techniques

Many pilots fly in a noisy cockpit and are sometimes using their radio at extreme distances between their transmitter and another receiver. For these situations, the FAA (Federal Aviation Administration) clearly defines in their 7110.65W how radio communication should be used by air traffic control. This order also governs weather reporting stations that will be informing pilots visa radio. These radio communication techniques and phraseology is put into place for the safety and efficiency of air traffic.

In general, when reporting numbers each number should be individually spoken. However, the exception to this rule is when the reporting number is in the thousands. Figures indicating hundreds and thousands in round number, as for ceiling heights, and upper wind levels up to 9,900 shall be spoken in accordance with the following, 500 pronounced five hundred 3,500 pronounced three thousand five hundred. Numbers above 9,900 shall be spoken by separating the digits preceding the word "thousand": 10,000 pronounced one zero thousand, 13,500 pronounced one three thousand five hundred. Up to but not including 18,000 feet MSL (Mean Sea Level), state the separate digits of the thousands plus the hundreds if appropriate. At and above 18,000 feet MSL (FL180), state the words "flight level" followed by the separate digits of the flight level: 19,000 pronounced Flight Level One Niner-Zero.

All directions communicated over radio are to be of a magnetic reference and not a true heading. Speed is to be reported in knots, and the word knots must be used after the

value of the speed has been spoken. The FAA also uses Coordinated Universal Time (UTC) for all operations. The word "local" or the time zone equivalent shall be used to denote local when local time is given during radio and telephone communications. The term "Zulu" may be used to denote UTC. When individually speaking letters the phonetic alphabet must be used. Overall, the goal of radio communication is to be as clear and concise as possible.

Information	Example Message Content	Non-Avionic Pronunciation	Avionic Pronunciation
Time	1321 EST	One - Twenty-One PM	One-Seven-Two-One Zulu or One-Tree-Two-One Local
	0239 EST	Two - Thirty-Two AM	Zero-Seven-Tree-Niner Zulu or Zero-Two-Tree-Niner Local
Wind Speed	35 Knots	Thirty-Five Knots	Tree-Five Knots
Wind Direction	90° True	East or 90°	Zero-Niner-Four Degrees
Thousands of Feet	11,500 Feet	Eleven Thousand Five Hundred Feet	One-One Thousand Five Hundred Feet
	20,000 Feet	Twenty Thousand Feet	Flight Level Two-Zero-Zero

Table 4.1.2: Phraseology Examples

4.1.3 METAR

METAR is a weather reporting format that is highly used in aviation. It is the most common format in the world for the transmission of observational weather data. This format has be standardized by the International Civil Aviation Organization (ICAO), which allows it to be standard throughout most of the world. A typical METAR will contain the ID of the weather reporting station, time in day of month and Zulu time, wind direction and speed (including gust), visibility, sky conditions, temperature, dew point, barometric pressure, and remarks. This format is used when reporting weather information over radio as well.

4.1.4 Traffic Advisory Practices Without Operating Control Towers

The Traffic Advisory Practices at Airports Without Operating Control Towers defines our project as an UNICOM system, under the guidelines that it is a “nongovernmental air/ground communication station which may provide information at public use airports.” In this standard is it stated that UNICOM stations can provide wind direction and wind speed information to pilots upon request, regardless if the UNICOM station shares the same operating frequency as the Common Traffic Advisory Frequency. This is important because in small airports which our project is aimed towards, will operate in the CTAF can commonly be assigned to a designated UNICOM frequency operating range. This is ideal for a small airport as the small amount of air traffic can be managed by commercial systems like our project, but in larger airport where the CTAF is different from the UNICOM frequency this can present itself a challenge as the pilot would have to switch between frequencies to communicate with the UNICOM system. This standard also calls for communication with UNICOM stations of at least 10 miles from the airport the station is in. This forces our system to be able to operate at such distances to comply with standards.

4.1.5 WAVE File

We will be using the WAVE format standard for storing audio data. The WAVE file standard was introduced as a joint standard from the IBM Corporation and the Microsoft Corporation in the “Multimedia Programming Interface and Data Specifications 1.0” standard document released in August of 1991. The WAVE file standard in particular was introduced as a substandard of the RIFF, or the Resource Interchange File Format, standard for storing multimedia. While old we chose this standard because it is the most common form of uncompressed audio, and is recognized across all systems as well as multiple audio centered programs. By using the WAVE format standard, we did not have to commit to a certain form of audio compression standard. This will allow us to directly interface with the raw audio data, as well as compress the data using any of form of audio compression standard in the future if we feel we need to compress the data.

The WAVE file format standard organizes the data it stores using what the RIFF standard defines as “chunks”. Each of these chunks, while having no particular set order to where they are located within the file, contain their own specific sets of fields and parameters. For the WAVE file format, the standards indicate that there are only three chunks that are required for any WAVE file; these three chunks include: the Header chunk, the Format chunk, and the Data Chunk. While there is no set order for these chunks, the adopted standard is to write each of the chunks in the order they were introduced above. This allows for readability, and the ability for programs to know where to look for certain

information without the need of including more header information about where data is located. This reduces the file size and the speed in which the file can be processed. Two optional chunks, the List chunk and the Info chunk, can be included in a WAVE file to document the order in which the various chunks appear in the current WAVE file. These two 10 chunks are usually place right after the Header Chunk and are only included for compatibility with software that did not follow the suggested chunk order adopted by the industry.

Each of the required chunks outline the basic needs of any multimedia player. The first is Header Chunk which specifies the multimedia format standard used by the file as well as the particular substandard of multimedia used. In the case of the WAVE format standard, the RIFF standard for multimedia, and the WAVE substandard are always included in the Header chunk. Along with these two fields the Header chunk contains the size (in bytes) of the rest of the file. The next required chunk, the Format chunk, is uses to specify the format in which the WAVE file was being recorded. Along with the standard chunk id and chunk size that outlines which chunk is being read and how large the chunk is, these fields are almost all variable and include the sampling rate, byte rate, number of channels, and bit resolution used to record the audio data. The only other major field to note that is included in the Format chunk is the Audio Format field which is used to specify what audio recording standard is being used to record the data. Because we are using an Analog to Digital converter to sample the audio we are recording, we will use the Pulse Code Modulation, or PCM, standard or audio recording. Lastly the WAVE file format standard requires the data chunk which is responsible for storing the raw audio data sampled in the audio format specified in the Format Chunk. This data is encoded in two's compliment format and then stored in the Little-Endian format.

4.1.6 Pulse Code Modulation

We will be using the Pulse Code Modulation audio format standard for recording audio data. This standard is used to digitally represent the analog audio data being recorded. We chose to use the PCM standard for recording audio data, as it directly coincides with how we will be receiving data from the analog to digital converter. The PCM standard requires taking a sample of an analog audio signal and representing it using a decimal number. Because most analog to digital converters use PCM to sample analog data, we will also be using this format.

4.1.7 I2C Standard

The Inter-integrated Circuit (I^2C) Protocol is a protocol intended to allow multiple "slave"

digital integrated circuits to communicate with one or more “master” chips. Like the Serial Peripheral Interface (SPI), it is only intended for short distance communications within a single device. Like Asynchronous Serial Interfaces, it only requires two signal wires to exchange information. I^2C is a protocol that was devolped by Philips Semiconductors in 1982 to be a simple bidirectional 2-wire bus for efficient inter-IC control. Only two bus lines are required: a serial data line (SDA) and a serial clock line (SCL). Serial, 8-bit oriented, bidirectional data transfers can be made at up to 100 kbit/s. Each device connected to the bus is software addressable by a unique address. It a true multi-master bus with included collision detection and arbitration to prevent data corruption. The I2C-bus is now the world standard that is currently implemented in thousands of different ICs, manufactured by many different companies.

I2C allows for simple, efficient communication between the sensors and the Raspberry Pi which makes it a good choice for our system. It simplifies how the software will poll from each sensor since the only thing that changes between weather sensors is the unique address.

These are just some of the benefits. In addition, I2C-bus compatible ICs increase system design flexibility by allowing simple construction of equipment variants and easy upgrading to keep designs up-to-date. In this way, an entire family of equipment can be developed around a basic model. Upgrades for new equipment, or enhanced-feature models (that is, extended memory, remote control, etc.) can then be produced simply by clipping the appropriate ICs onto the bus. If a larger ROM is needed, it is simply a matter of selecting a microcontroller with a larger ROM from our comprehensive range. As new ICs supersede older ones, it is easy to add new features to equipment or to increase its performance by simply unclipping the outdated IC from the bus and clipping on its successor.

Designers of microcontrollers are frequently under pressure to conserve output pins. The I 2C protocol allows connection of a wide variety of peripherals without the need for separate addressing or chip enable signals. Additionally, a microcontroller that includes an I 2C interface is more successful in the marketplace due to the wide variety of existing peripheral devices available.

The possibility of connecting more than one microcontroller to the I2C-bus means that more than one master could try to initiate a data transfer at the same time. To avoid the chaos that might ensue from such an event, an arbitration procedure has been

developed. This procedure relies on the wired-AND connection of all I2C interfaces to the I2C-bus. If two or more masters try to put information onto the bus, the first to produce a 'one' when the other produces a 'zero' loses the arbitration. The clock signals during arbitration are a synchronized combination of the clocks generated by the masters using the wired-AND connection to the SCL line

Generation of clock signals on the I2C-bus is always the responsibility of master devices, in this case, the Raspberry Pi. Each master generates its own clock signals when transferring data on the bus. Bus clock signals from a master can only be altered when they are stretched by a slow slave device holding down the clock line or by another master when arbitration occurs.

4.1.8 Python Standards

Since Python is our language of choice, there are a few standards within the language we need to adhere to so that the code compiles correctly and so that the code can be maintained and is easily understandable. PEP8 is the style guide written by Python Software Foundation which serves as the official documentation for the language.

The style guide helps enforce consistency. Consistency with this style guide is important. Consistency within a project is more important because it makes the code easier to read and thus easier to maintain. Consistency within one module or function is the most important because this way the function will compile correctly and perform the task you expect it too.

Continuation lines should align wrapped elements either vertically using Python's implicit line joining inside parentheses, brackets and braces, or using a *hanging indent*. When using a hanging indent there should be no arguments on the first line and further indentation should be used to clearly distinguish itself as a continuation line. This is important to keep in mind because as the code gets more complex or lengthy (like in the case of the text to speech sections) the ability to wrap lines of code makes it much easier to read. In addition, Python is very picky about indentation. New lines are specified by indents instead of the semi-colon on the previous line like many other languages. In order for the code to compile correctly, each line has to be indented the correct number of times in order to match up with braces and conditional statements. This includes any wrapped text.

When the conditional part of an if -statement is long enough to require that it be written across multiple lines, the combination of a two character keyword (i.e. if), plus a single

space, plus an opening parenthesis creates a natural 4-space indent for the subsequent lines of the multiline conditional. This can produce a visual conflict with the indented suite of code nested inside the if -statement, which would also naturally be indented to 4 spaces. This takes no explicit position on how (or whether) to further visually distinguish such conditional lines from the nested suite inside the if -statement. The closing brace/bracket/parenthesis on multi-line constructs may either line up under the first non-whitespace character of the last line of list. Again, it is crucial for any piece of Python code to have the correct amount of spaces or indentations before each line. This was a very important aspect of this system because it is easy to misalign text which would have caused the system tests to fail.

Spaces are the preferred indentation method and tabs should be used solely to remain consistent with code that is already indented with tabs. Python 3 disallows mixing the use of tabs and spaces for indentation. Python 2 code indented with a mixture of tabs and spaces should be converted to using spaces exclusively. This is another language specific issue that Python poses because it is such a finicky language.

When invoking the Python 2 command line interpreter with the -toption, it issues warnings about code that illegally mixes tabs and spaces. When using -tt these warnings become errors. These options are generally recommended because it allows us to verify the code before deploying it.

Limit all lines to a maximum of 79 characters. For flowing long blocks of text with fewer structural restrictions (docstrings or comments), the line length should be limited to 72 characters. Limiting the required editor window width makes it possible to have several files open side-by-side, and works well when using code review tools that present the two versions in adjacent columns. This helps with readability and allows faster debugging which is very important for testing but also for maintenance. Just like any system, ours will need to be periodically maintained to keep up with updating software and security standards. This system should not be vulnerable to any external threats so to mitigate that risk, the first step is for the code to be easily maintained.

The default wrapping in most tools disrupts the visual structure of the code, making it more difficult to understand. The limits are chosen to avoid wrapping in editors with the

window width set to 80, even if the tool places a marker glyph in the final column when wrapping lines. Some web based tools may not offer dynamic line wrapping at all. Some teams strongly prefer a longer line length. For code maintained exclusively or primarily by a team that can reach agreement on this issue, it is okay to increase the nominal line length from 80 to 100 characters (effectively increasing the maximum length to 99 characters), provided that comments and docstrings are still wrapped at 72 characters.

The Python standard library is conservative and requires limiting lines to 79 characters (and docstrings/comments to 72). The preferred way of wrapping long lines is by using Python's implied line continuation inside parentheses, brackets and braces. Long lines can be broken over multiple lines by wrapping expressions in parentheses. These should be used in preference to using a backslash for line continuation.

Surround top-level function and class definitions with two blank lines. Method definitions inside a class are surrounded by a single blank line. Extra blank lines may be used (sparingly) to separate groups of related functions. Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations). Use blank lines in functions, sparingly, to indicate logical sections. Python accepts the control-L (i.e. ^L) form feed character as whitespace; Many tools treat these characters as page separators, so you may use them to separate pages of related sections of your file. Note, some editors and web-based code viewers may not recognize control-L as a form feed and will show another glyph in its place.

For Python 3.0 and beyond, the following policy is prescribed for the standard library (see PEP 3131): All identifiers in the Python standard library MUST use ASCII-only identifiers, and SHOULD use English words wherever feasible (in many cases, abbreviations and technical terms are used which aren't English). In addition, string literals and comments must also be in ASCII. The only exceptions are (a) test cases testing the non-ASCII features, and (b) names of authors. Authors whose names are not based on the latin alphabet MUST provide a latin transliteration of their names.

In order for this system to meet Python standards, they had to be taken into consideration from the very beginning of the logic planning and writing process. These standards help to make the code easily understandable and easily maintained which are crucial for any

system. All of the standards noted above were instrumental in making the code as easy to read as possible in addition to ample comments detailing what each block of code is meant to accomplish.

4.1.9 Django Standards

Django is a web framework that combines HTML, Python, and SQL to easily, quickly, and efficiently create websites and databases with logic that can easily morph and scale. The premise behind it is to increase turn around and allow projects to be created much more quickly. The framework is consists of models, views, and templates. Models are the format of the databases such as the column headers, types of data, and any data constraints. Views are the logic for each webpage or function of the website/app. Templates are the web page itself with the special markers specifying which sections of code are actual HTML and which sections are Python and need to be interpreted.

The important standards of this framework that must be followed in our implementation are the set up of the models, views, and templates, the special syntax denoting which sections of code in the template are Django and need to be translated or executed from Python to HTML, and the file structure so that Django knows where to look for certain files.

It is important for the Django framework to be set up correctly on the machine that will run it because there are many files that Django expects to exist in certain places so it is crucial those conventions be followed.

4.2 Design Constraints

In this section, we will talk about the different realistic design constraints we will encounter when tackling this project. We will discuss various things from time constraints, budget constraints and other related real-world constraints we might encounter.

4.2.1 Time Constraints

This project will be a complete working product by the end of Senior Design II in Summer 2017. This creates a limited timeframe for the team to work with. The total time for this project is about 28 weeks, and to develop, design, build, and test a system of this nature will take diligence to complete in that amount of time. The plan was to have a working prototype at week 11, at the end of Senior Design I. This in and of itself was a lofty goal and requires teamwork and persistent hard work.

Due to the time constraints it was crucial for us to have a working task list that detailed every aspect of the system and when it was to be completed by. Though some of the tasks fell a bit behind, in the end it was possible to complete the system and have a working finished project for the end of Senior Design II.

Like with any project, you are never truly finished. Though we were able to complete a working version of the system, there is still room for improvement and the ultimate test is whether Professor Michael Young decides it meets all of his requirements and if he decides to deploy it in his hangar at the Orlando-Apopka airport.

4.2.2 Budget Constraints

The team is comprised of four college students with limited incomes, which limits the solutions, but also provides motivation to make this as low-cost of a system as possible. Our primary sponsor has provided us with $250 towards our project and that has been set as the target cost for the entire system. If the need arises, the team can use up to $500 before having to use personal funds. This provides us with a good financial base to build our project on, but without having unlimited funds, the team will have to be mindful of the limited budget.

This serves as a guarantee for a cost-effective solution which was achieved. The total spent on this project was just over $700 which was due to ordering spare parts. Once that total is broken down and itemized to the parts it took to complete one fully-functioning system, we were well below our goal at around $400. The budget is detailed further on in section 7.2 of this document with an itemized list of what was purchased versus what was used to complete one working model of the system.

5. Design

This chapter covers both the hardware and software design of the Auto FBO system. The hardware design is covered first followed by the software design.

5.1 Power Supply Design

Shown in table 5.1 are all the components used in the Auto FBO system along with their needed supply voltages and max or recommended currents. A miscellaneous category under the components has been considered in the design to account for more components that will be potentially be incorporated, as well as those not listed that are included in the actual design. The total max current demand of this design is estimated to be 4.8 A, along with supply voltages of 3.3, 5, and 15 V.

Component(s)	Supply Voltage (V)	Max or Recommended Current Supply (A)
Raspberry Pi 3B	5	2.5
Radio	11.04 -15.87	1
Operational Amplifiers	10 -18	0.05
Anemometer	3.3	0.005
THD Sensor	3.3	0.005
CODEC	3.3	0.200
ADC	3.3	0.001
Comparator	15	0.050
Miscellaneous	N/A	1

Table 5.1: Power Supply Demands

Since current AC to DC power supply modules are relatively cheap and easily accessible, an AC to DC power supply module will act as the central power supply unit. Branching from this supply are voltage regulators to provide the necessary supply voltage rails needed for the system. A block diagram of the power supply system is shown below. This

design approach is taken in respect to cost, efficiency, and voltage noise and ripple, as well as simplicity of the design.

Most of the commercially available power supply units, which supply high power, are switch mode power supplies. These supplies are highly efficient that can reach efficiencies above 90%. Since all the power of the Auto FBO system will be transferred through the central power supply unit and then distributed to the various regulators, it is necessary that it be a switch mode power supply. However, switch mode power supplies do present a high margin of voltage ripple and noise. The unwanted effects from the central power supply will be dampened by the linear voltage regulators.

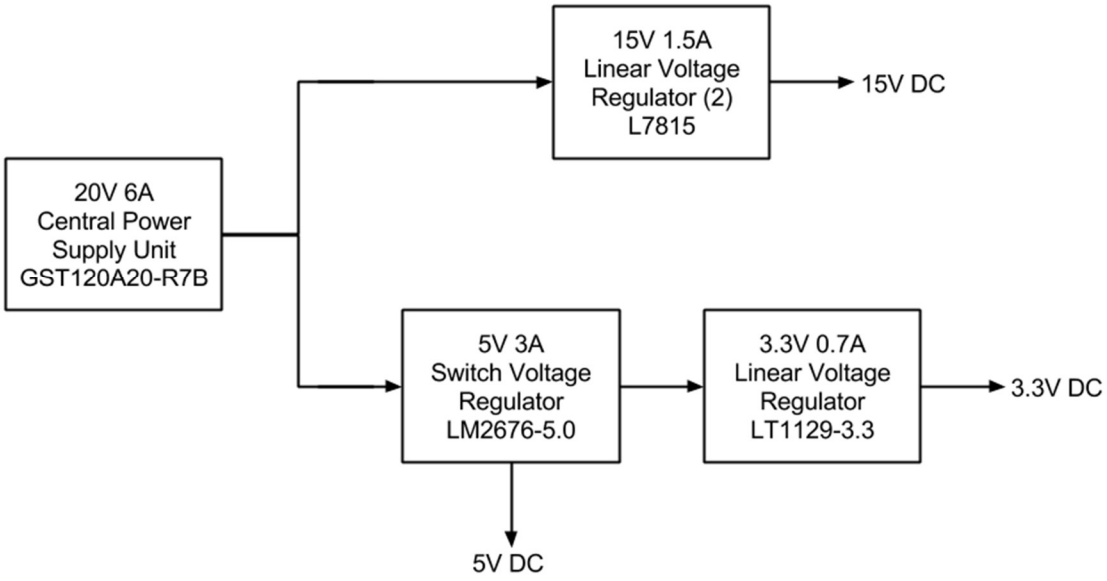


Figure 5.1: Power Supply Unit Block Diagram

5.1.1 Voltage Regulation

The usage of the linear voltage regulators are chosen not only to help block unwanted characteristics of the switch mode central power supply unit, but also to provide the necessary various voltages that the Auto FBO system requires. Linear regulators have very low output voltage ripple because there are no elements switching on and off frequently, and linear regulators can have very high bandwidth. Furthermore, linear regulators are simple and easy to use, especially for low power applications with low output current where thermal stress is not critical. These characteristics are critical to the needs of this power supply for supplying power to communication and audio components

in this system and providing a simple design solution.

5.1.1.1 3.3V Regulator

Power is supplied to the input pin to the LT1129I. This power will be received from the LM2676-5.0 switching voltage regulator to efficiently create a 3.3V rail by creating a lower voltage drop across the device. This input voltage is acceptable for the regulator since it has an absolute maximum input voltage rating of 30V and a low dropout voltage of 400 mV. According to the datasheet, “the input pin should be bypassed to ground if the device is more than 6 inches away from the main input filter capacitor. A bypass capacitor in the range of 1μF to 10μF is sufficient. The LT1129 is designed to withstand reverse voltages on the input pin with respect to both ground and the output pin. In the case of a reversed input, which can happen if a battery is plugged in backwards, the LT1129 will act as if there is a diode in series with its input. There will be no reverse current flow into the LT1129 and no reverse voltage will appear at the load. The device will protect both itself and the load.” The output pin supplies power to the load, and is recommended to use an output capacitor at the output to prevent oscillations. The minimum recommended value is 3.3μF with an ESR of 2Ω or less. The shutdown pin, SHDN, is used to put the device into shutdown if it is actively pulled low. According to the datasheet, “if the shutdown pin is not used it can be left open circuit. The device will be active, output on, if the shutdown pin is not connected.” The fixed voltage version of the LT1129I used for this design uses the sense pin as an input to an internal error amplifier. The sense pin can be directly connected to the output pin, or at the load if better regulation is needed.

5.1.1.2 5V Regulator

The input pin of the LM2676-5.0 is supplied power by the 20V central power unit to regulate a fixed output voltage of 5V at the output pin. The input voltage of 20V from the central power unit is acceptable since the device has an absolute maximum input voltage of 45V. The output circuitry of this regulator was designed with guidance from the LM2676-5.0 datasheet. The 100 μF capacitors are used to smooth the switched DC output voltage and provide energy storage for peak supply demands. The 33 μH inductor was chosen to efficiently store energy during the on-switch time and transfer its stored energy during the off-switch time. The 1N5822 catch diode provides a current flow path when during the off-switch time, when the current through the inductor continues to flow. During this time, the diode is forward biased and clamps the switch output to a voltage below ground. The efficiency of the supply is significantly impacted by the power loss in

the diode. During the on-switch time the diode is reversed biased. “The boost capacitor creates a voltage used to overdrive the gate of the internal power MOSFET. This improves efficiency by minimizing the on resistance of the switch and associated power loss.”

5.1.1.3 15V Regulator

The input pin of both L7815 is supplied power by the 20 V central power unit to regulate a fixed output voltage of 15V at the output pin. The input voltage of 20 V from the central power unit is acceptable since the device has an absolute maximum input voltage of 35V. According to the datasheet, “it is recommended that the regulator input be bypassed with capacitor if the regulator is connected to the power supply filter with long lengths, or if the output load capacitance is large. An input bypass capacitor should be selected to provide good high frequency characteristics to insure stable operation under all load conditions. A 0.33μF or larger tantalum, mylar or other capacitor having low internal impedance at high frequencies should be chosen.”

5.1.2 Overall Power Supply Design

Shown in the figure below is the power supply design for the Auto FBO system. The central power supply unit (CPSU) supplies 20V to all four linear voltage regulators. The line to the regulators also contains shunt electrolytic capacitors. These capacitors are included for several reasons including recommended application suggestions of the datasheets, increased capacitance, low ESR, high frequency impedance, reliability, redundancy, and peak current demands. The output of each regulator also includes shunt electrolytic capacitors for the same reasons. KEMET 49X tantalum capacitors were used for the input and output capacitors to aid in the design in respect to the characteristics above.

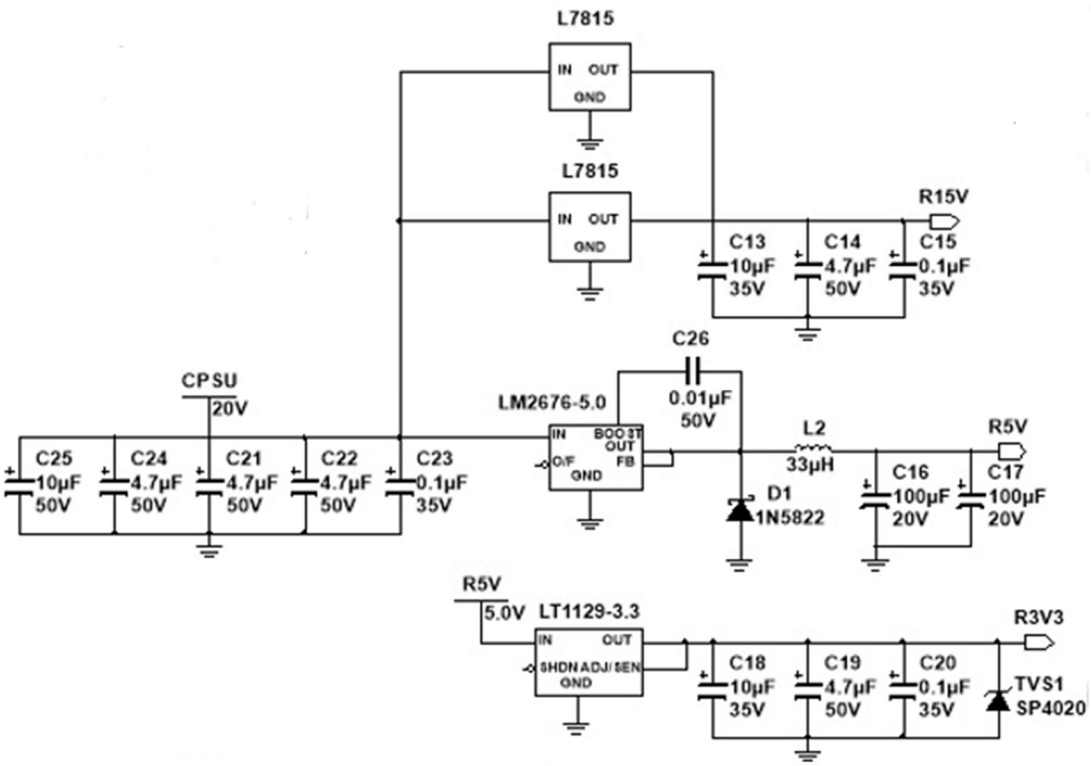


Figure 5.1.1: Power Supply Design Schematic

5.2 Interface Board Design

The interface board will interpret all incoming and outgoing signals between the radio and the Microprocessor. This will be handling the TX and RX signal conditioning, conversion and amplification between the two systems. This will also push the PTT signal into the radio for whenever a transmission is going to be sent out to the pilot requesting information. Inputs will be received from directly tapping into the radio at specific solder points or through the back pins of the IC-A2 VHF Aircraft radio. In this the communication between the UNICOM programmable HUB, the Raspberry Pi 3, and the broadcasting hardware, the IC-A2 VHF Aircraft radio.

5.2.1 PTT Circuit

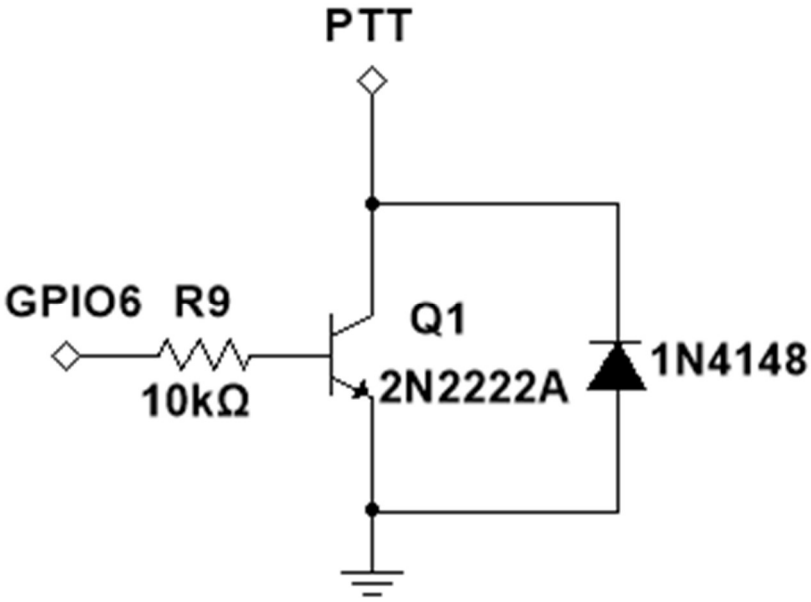


Figure 5.2.1 PTT Circuit

To communicate to the Raspberry Pi 3's intent to transmit a signal needs to be pushed so the IC-A2 Radio in order to get it in a 'Ready to Transmit' state. This signal is going to be generated by the Raspberry Pi 3's GPIO pin and a DC power source for system testing. This will require two inputs: one for system use and one for system troubleshooting. The input from the Raspberry Pi 3's GPIO pin will be for practical use, thus the DC voltage source will be used for testing. During testing the GPIO pin will act as a ground and part of the current will be sent through there and the rest will be sent to the PTT input of the radio. This will allow the user to check if the circuit is bad or if there has been a programming error in the Raspberry Pi 3 system. The intent of this circuit is to simulate the PTT signal generated by the microphone interface in the radio. The idea is to act grounded when not transmitting and to input a current when ready to transmit in order to open the mic channel and set the IC-A2 in a ready to transmit mode.

The Push-To-Talk (PTT) circuit is going to be responsible for setting the IC-A2 radio into transmit mode. This is done by using the GPIO pin in the Raspberry Pi 3's pins as a 3.3V source. This voltage being pushed through the NPN transistor, Q1, pulls the PTT relay day to ground. The action of pulling the relay to ground results in the collapse of the magnetic field around the inductor. This will send a large voltage back from the PTT relay

to the Q1 transistor. This is where the reverse biased diode will re-route that voltage to ground, thus not burning the transistor. When it comes to testing the system switch, S1, will have the 3.3V source from the interface board act as the Raspberry Pi 3's GPIO input. This will simulate the act of readying for transmit on the IC-A2. Though the design shows the GPIOPIN power source as a 3.3V power source it must be noted that this is a pin from the Raspberry Pi 3's interface. This will act as a ground when being tested as the system will be inactive, or turned off, when being tested. The circuit takes full advantage of the Raspberry Pi 3's architecture to reduce the number of components required to achieve the same function. When using the Raspberry Pi 3's GPIO as a ground its current limits is around 16mA maximum current before burning the microprocessor. Therefore, the current running from the interface board power supply is split using resistors R1 and R2 above.

5.2.2 Carrier Detect

The consolidation between the Radio and Interface Board serves as the bridge to be able to condition the carrier detect and identify when there will be transmission. Since we only have two (2) levels for identification, a comparator is being used to compare and determine which level, that indicates transmission or no transmission, is being received.

The comparator being used is the LM393 Dual Differential Comparator. The purpose of this device is to compare two (2) voltage values, and output a digital signal indicating which of the two is larger to the main control unit through a GPIO.

The differential comparator consists of a high gain differential amplifier. These devices are commonly used in systems that measure and digitize analog signals such as analog to digital converters, as well as relaxation oscillators. In our application, we compare the received signal, carrier detect present or carrier detect not present, with a reference voltage.

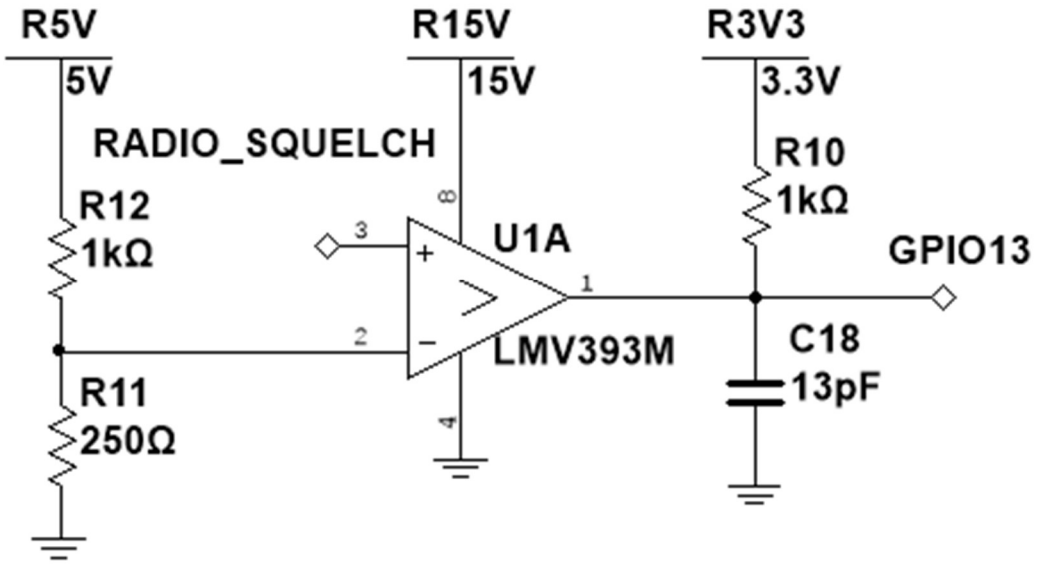


Figure 4.5 Comparator Circuit

The voltage measured for RX audio signal (CD) being present was 1.4V; meaning, when compared to the reference voltage, 1V, the comparator will output a logical 1, allowing the 3.3V become the output to the next stage of the circuit. Next stage of the circuit being to a GPIO pin of the microcontroller. The voltage measured for RX audio signal (CD) not present was 0V; meaning, when compared to the reference voltage, 1V, the comparator will output a logical 0, this output will not allow the 3.3V become the output to the GPIO pin. See image above.

$$V0 = \begin{cases} 0, & V+ < V- \\ 1, & V- \geq V- \end{cases}$$

The 1V for reference are achieved through a voltage divider circuit. The input (Vin-) is 5V which is then divided through both resistors of 1k ohms and 250 ohms. The reference is then then compared to the ground at the 1k ohms resistor. This will create a constant output of 1V since the 5V is being provided by a voltage regulator.

5.2.3 RX Buffer Audio Design

While testing the radio with the CODEC it was found that the CODEC was severely loading the radio when it was transmitting audio to the CODEC while the CODEC was recording. To negate this problem a buffer was inserted between the audio signal coming out of the radio and the input of the CODEC.

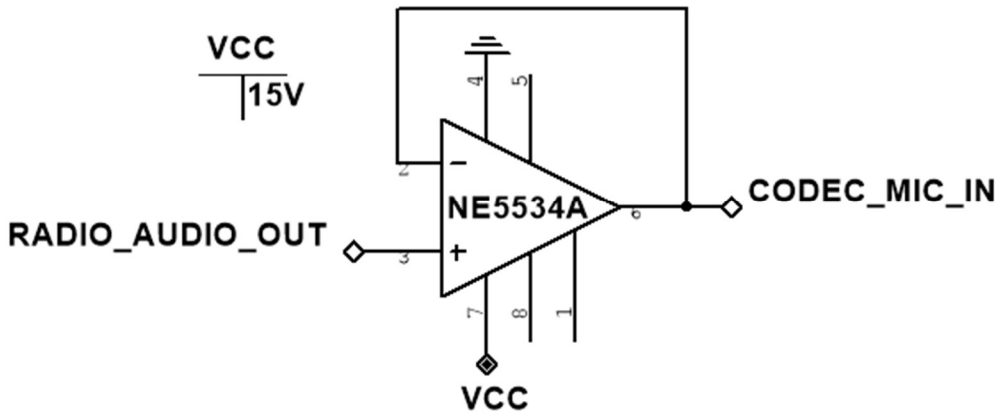


Figure 5.2.3: Rx Buffer Audio Design

5.2.4 TX Filter and Bias Audio Design

In transmitting audio out from the Raspberry Pi then to the CODEC we found that there was high frequency noise being produced. This lead to the decision of a low pass filter being needed. This was done by using a second order low pass Butterworth filter with a cutoff frequency of 50 kHz. The Butterworth filter was chosen as it can provide a maximally flat passband which is needed as to not alter the audio signal. The cutoff frequency of 50 kHz was chosen since it is known that audio signals range from 0-20 kHz, and to ensure that as the passband started to drop off near the cutoff frequency it would produce negligible difference between upper audio signals.

Since the filter design is a 2nd order Butterworth the denominator of the transfer function is $s^2 + \sqrt{2}s + 1$. This sets $\frac{\omega_0}{Q} = \sqrt{2}$ with $Q = \frac{\sqrt{2}}{2}$. This Q value is desirable for this design as to not create a rise in gain as the cutoff frequency is approached. Using frequency scaling k_f was set to 2π×50000 to set the cutoff frequency at 50 kHz. $C'_1$was set to 200 pF to set to C'_2 100 pF so that these capacitors could be commercially bought as these values are common. Solving for the magnitude scaling factor, k_m sets R' to 22.508 kΩ, which will be implemented with commercially available 47 kΩ and 43.2 kΩ resistors in parallel.

$$C'_1 = \frac{1}{k_m k_f}\frac{2\sqrt{2}}{2} = 200 \text{ pF}$$

$$C'_2 = \frac{1}{k_m k_f}\frac{\sqrt{2}}{2} = 100 \text{ pF}$$

$$R' = k_m = 22.508 \text{ kΩ} = 43.2 \text{ kΩ} \,||\, 47 \text{ kΩ}$$

Before this filter is a decoupled inverting amplifier circuit network of unity gain which sets an offset of 7.5 V since the operational amplifiers are set between 15 V and ground. Without this network, the audio signal could potentially be cut off. The resistor divider biasing technique is low in cost and keeps the op-amp's dc output voltage at halfway between the supply voltage, however the operational amplifier's common mode rejection still depends on the RC time constant formed by RA||RB and capacitor C2. Using a C2 value that provides at least 10 times the RC time constant of the input RC coupling network (R1/C1) will help insure a reasonable common-mode rejection ratio. With 100 kΩ resistors for RA and RB, practical values of C2 can be kept small if the circuit bandwidth is not too low. Depending on the supply voltage, typical values that provide a reasonable compromise between increased supply current and increased sensitivity to amplifier bias current, range from 100 kΩ for 15V or 12V single supplies.

Considering the characteristics of this decoupled inverting amplifier circuit network of unity gain RA and RB were set to 100kΩ with R1=R2 to achieve unity gain as well as minimize input bias current errors by keeping R2 one-half of RA. The input and output capacitors are selected to be 40μF to achieve a low impedance for low frequency audio signals. The bypass capacitor C2 was chosen to be 470 μF to help insure a reasonable common-mode rejection ratio and unity gain.

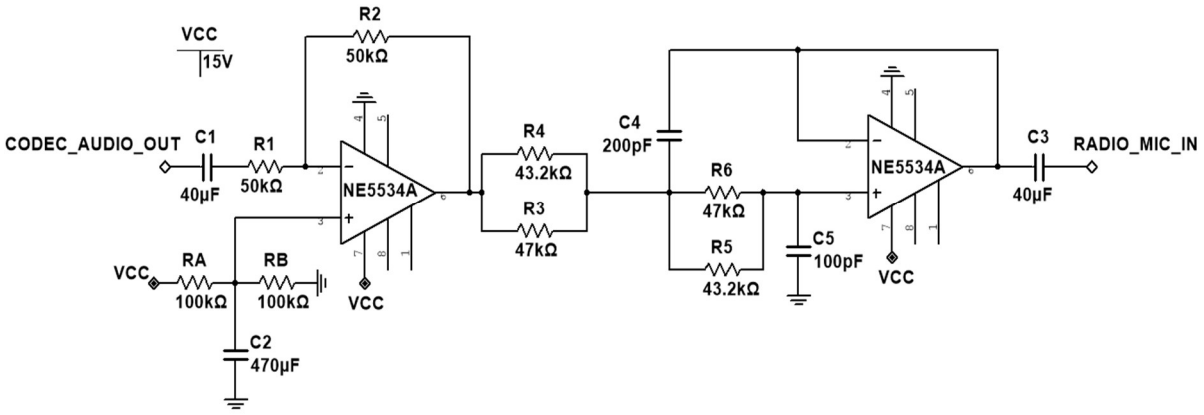


Figure 5.2.4: Tx Filter Audio Design

5.2.5 Anemometer and Wind Vane Design

The Davis Instruments 7911 Anemometer uses a RJ11 4P4C as an interface to communicate to external devices. This interface is composed of four wires connected to specific components within the sensor as shown on the right side of figure 5.2.5.2. The yellow wire is used to supply the 20 kΩ potentiometer. This potentiometer is also connected to the green wire that is used to indicate the wind direction. The reed switch is used to compute the wind speed and is connected to the black and red (ground) wires.

Internally, both the potentiometer and reed switch are used to sense wind speed and direction. Wind speed is measured by the opening and closing of the reed switch, which is connected to ground. Each revolution of the anemometer wind cups caused the switch to open and close. This action is implemented by a magnet coming in close proximity to the switch as the cup mechanism is rotated. When the magnet is brought into close proximity to the reed switch the internal leads close. Conversely, when the magnet moves away from the reed switch the leads open. Wind direction is measured by a circular 20 kΩ potentiometer. Depending on the direction of the fin, the wiper of the potentiometer is moved. As shown in figure 5.2.5.1 this potentiometer has a “dead zone” where the wiper makes no contact.

The design of our wind sensor interface compared to the previous group’s design significantly reduces the amount of components, power, and provides more accurate data. Their design included a BJT transistor, 6 resistors, and a LED, while our design only uses 3 resistors and a LED. Their wind speed design used a transistor with resistors to create a voltage controlled switch, which is not needed since the reed switch in the instrument already performs this function. Not only does this use excess components, but also uses more power with the same result. Their wind direction design uses a voltage divider, which was also not needed as they could have only supplied 3.3 V to the anemometer and used no divider. This division also neglects to fully suppress the “dead zone” in the potentiometer.

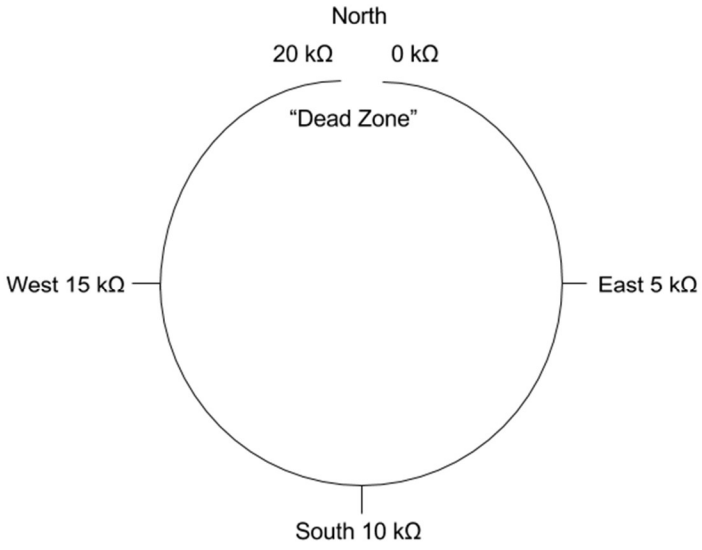


Figure 5.2.5.1: Wind Potentiometer

Shown on the left side of figure 5.2.5.2 is the interface design for the Davis Instruments 7911 Anemometer. A 10 kΩ resistor is used after the reed switch to reduce the amount of current through the reed switch when it closes to ground. The reed switch and the 10 kΩ resistor connected to 3.3 V provides an active low pulse from 3.3 to 0 V to the RPI GPIO when the cups of the anemometer makes a revolution. The 20 kΩ is used in conjunction with the wind direction potentiometer to fill in the "dead zone". Once the wiper of the potentiometer falls in the "dead zone" where no contact is being made the 20 kΩ resistor provides a transition between the wiper making contact on the 20 kΩ side and the 0 kΩ side. The ADC will receive a voltage range of 0 to 3.3 V depending on the wiper's position. The LED is included to show that the wind sensor is receiving power and is providing data to the RPI and ADC.

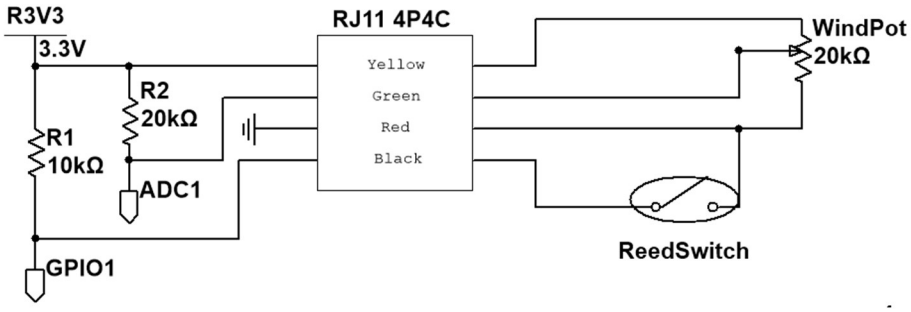


Figure 5.2.5.2: Anemometer Interface Design

5.2.5.1 Analog to Digital Converter

After the AGC voltage is received and conditioned it goes to the analog to digital converter. The Analog to digital converter chosen was the ADS1015.

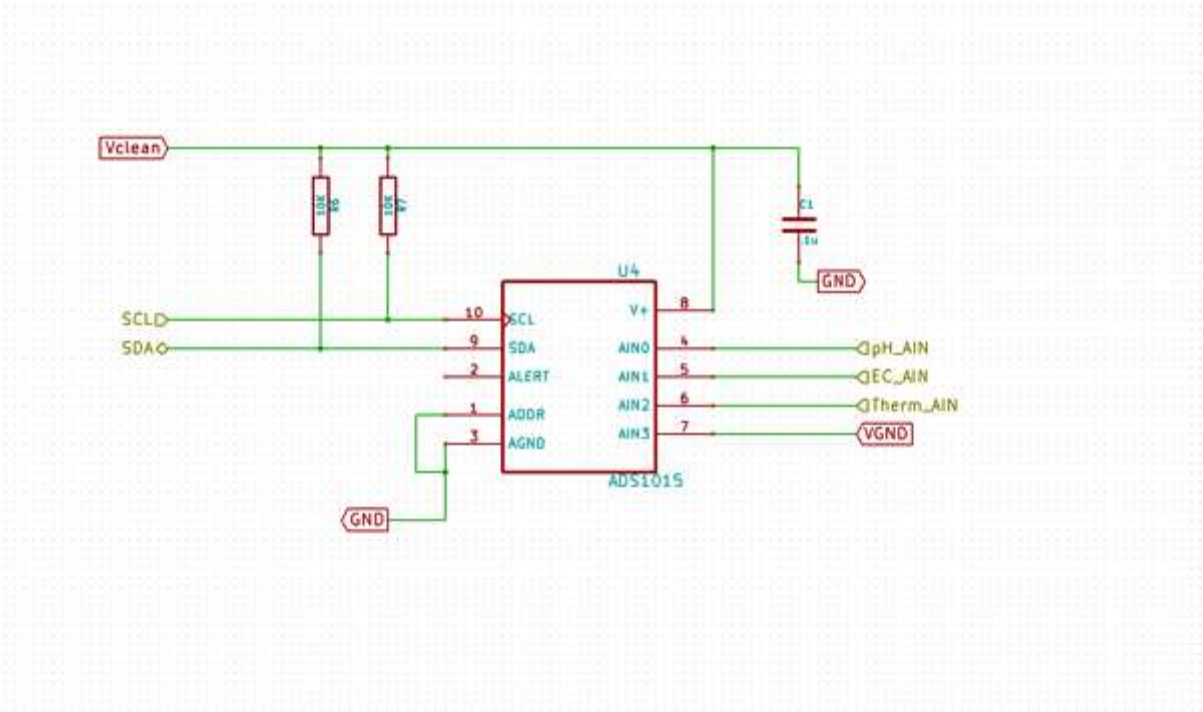


Figure 4.2.5.1 ADS1015 Application Circuit

The purpose of the analog to digital converter (ADC) is to provide the microcontroller with a digital number that is proportional to the magnitude of the signal, voltage or current, sent from the AGC. The conversion of this signal involves some error parameter. The higher the number of bits, resolution, available on the ADC, the more precise the conversion can be. The ADS1015 allows a precision of 12 bits, this indicates the number of discrete values it can produce over the range of analog values. An ADC is defined by the bandwidth available, range of frequencies, and its signal to noise ratio.

5.2.6 I2C Bus

The ADS1015 converts the analog signal to digital signal with a precision range of 12 bits. The signal is then delivered to the Raspberry Pi through this I2C Bus. The I2C Bus is able to communicate to a multitude of other peripheral devices (defined as “slaves”) by assigning a unique address to each device. The Raspberry Pi is considered the “master”

device and retains the right to read and write to the incoming signals. The other devices on the I2C bus, the slave constructs, require explicit permission from the Raspberry Pi in order to read and write. The I2C bus can support well over 1000 devices using only two lines -the SDA and SCL lines. For this reason, and also because it is less messy than the SPI connection configuration with the GPIO pins, it works quite perfectly for our system.

For our system, the anemometer and the temperature/pressure/humidity sensor will communicate with the Raspberry Pi through the I2C bus. They each have a unique address on the bus which will all the software on the Pi to reach them individually to poll for the current weather conditions.

5.2.7 PCB Design

Using Eagle, we were able to layout the PCB while taking into consideration a number of design guidelines. For example, care has been taken to place all bypass caps as close to the IC's as possible; same goes for the feedback loops on our op-amps, all feedback capacitors are placed as close as possible to the noninverting pin. Also, all digital signals are segregated to the right side of the board and all analog signals to the left of the board; this helps to limit digital noise interfering with the analog processing. Included below is a picture of our final PCB design with the top layer in red and the bottom layer in blue.

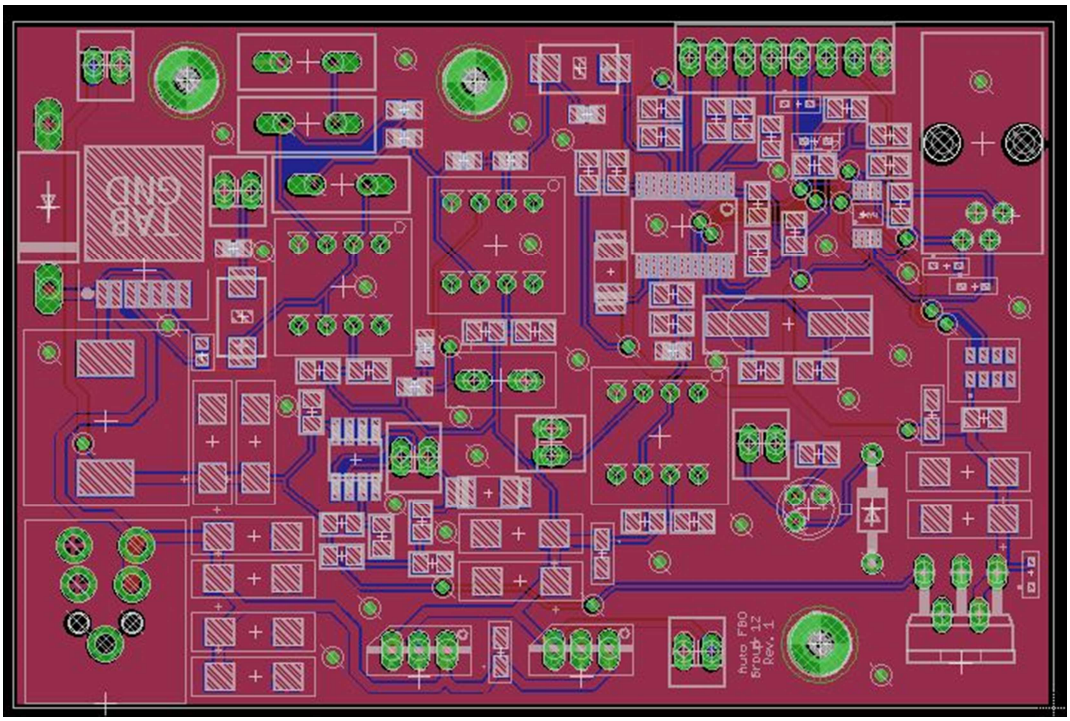


Figure 5.2.7 Final PCB Design

5.3 Software Design

This section details the software logic behind the system. This system is broken down into two main programs, the main logic loop program and the weather polling program. These two programs work together to detect carrier signals and respond to them. When the pattern recognition function matches with the carrier detected click-pattern, the system then decides on an action - whether to announce the current weather condition to the user or proceed to a communications check. Said action is then performed in a timely manner within a few seconds since the pilot would need to receive the requested current wind conditions on his way to land. The program that collects the weather data is separate from the main loop so that there is less of a delay in the carrier detect and so that the weather measurements can be wrapped up neatly in an object. Creating a weather object allows the program to easily pass the measurements back to the main loop so that it can concatenate the audio file to stream back to the pilot.

As mentioned in Chapter 3 our software will be running solely on the Raspberry Pi in the Python language. The code will utilize the Django framework for the database aspect of the software. This will require a model for the database structure. This model will include aspects of wind conditions that should be saved. These attributes include date and time, wind direction, wind speed, variable wind conditions if detected, and wind gust if detected, but this list can be expanded in the future.

The Django framework allows us to have a way to store previous weather information and it allows us to easily create the web interface which will be used to remotely access the weather conditions and for the administrator of the system to change certain parameters.

5.3.1 Main Logic Loop

This is the main loop for this system's software. After the Raspberry Pi is powered on, it will automatically launch the main program. After the main program is started, it will begin an initialization process. This process includes starting the separate weather program and making sure it is operational and responsive, then it will also start the webserver. The separate weather program will poll the sensors for wind speed, direction, temperature, humidity, and pressure and when called upon, it will return an object will the most current values for each weather condition.

We chose to separate this into its own program because it allows the main program to

handle the carrier detect more efficiently, allows us to easily compute the wind speed and direction values, and it will also allow us to set intervals for how often we want certain weather conditions to be read or computed without overcomplicating the main loop. It will be much more efficient to receive an object with all the weather readings in the main program instead of having to poll each sensor when the information is requested. Polling each sensor when the weather is requested would result in a delay of when the synthesized audio would play back to the pilot. This is due to the nature of some of the sensors and the measurements being read. In order to report wind speed and direction, the values have to be calculated by recording values from the anemometer over a period of time and then finding the average. In addition, the temperature/pressure/humidity sensor has a delay of a couple of seconds while it takes its measurements.

After the initialization process, the main program will begin to listen for a carrier signal. When a carrier signal is detected, the program will enter a function to count the clicks which is described in detail in section 5.2.2 of this document. After the clicks are detected and a decision is made as to whether the pilot is requesting the weather or a communications check, the main loop will jump into either function and perform the needed action. Both of these functions will be described in greater detail in the following sections.

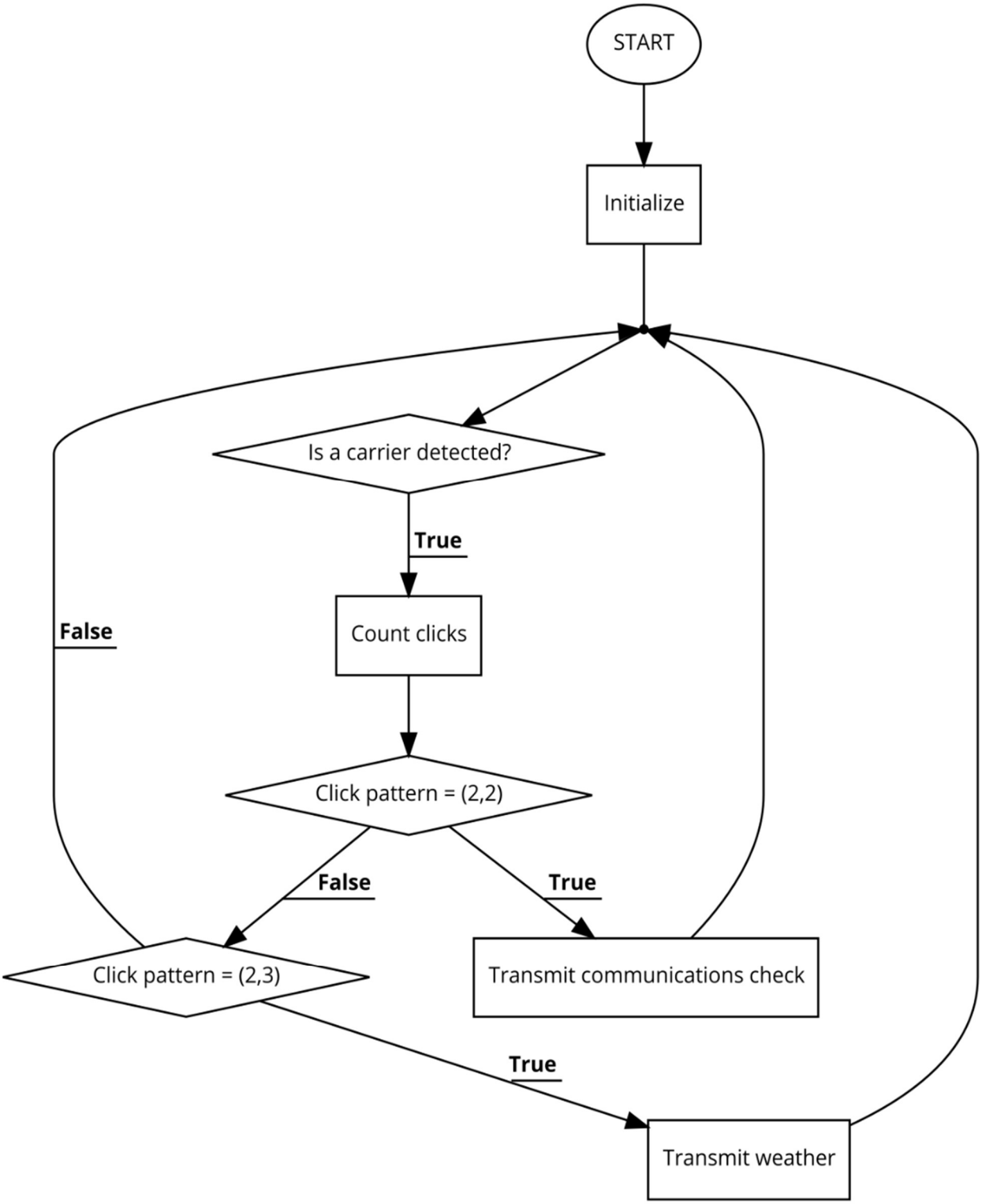


Figure 5.2.1 Main Logic Loop

Now that the software has counted the number of clicks, it will compare the pattern it has found to the patterns needed to request the current weather conditions or a communications check. If the clicks counted adhere to the pattern of two clicks followed by a pause and then two more clicks, then the program will enter a function to transmit a radio check which is described in detail in section 5.1.5 of this document. If the pattern detected is two clicks followed by a pause and then three more clicks, then the program will enter a function to transmit the current weather report which is described in detail in section 5.1.4 of this document. If the clicks detected match neither pattern, then the program will ignore the clicks and return to listening for a new carrier signal. This last bit is important because the system needs to always look for a carrier signal. There is no sense in continuing to try to detect a pattern if any one segment of the pattern is not within the maximum and minimum parameters set in the count clicks function. The administrator of the system for each airport will have the ability to change the maximum and minimum parameters since they need to have the ability to change the click pattern to avoid system conflicts.

In practice, our main logic loop worked out slightly different than we had originally planned. We still had to define the values for the gap, dwell, and on times for the carrier detect logic but we decided those should be modified by the administrator and coded them in such a way that the values are pulled from the web interface and if there is no value in the web interface, the program will operate with a default pattern.

Next, the main program calls the weather polling function to collect the current weather conditions. These results are stored in an object which keeps all the readings from the same point in time collected and makes them accessible by attribute name.

From there, we enter the carrier detect logic. Here, the code calls the carrier detect function to try to find the correct pattern. It will listen and check the timing of each signal against the dwell, on, and gap times previously mentioned. If the signal received does not match up with the timing for the next phrase of the pattern, the carrier detect loop will restart, ignore the previous signals, and start to listen for a new carrier signal. Once the carrier detect function has found either the weather or transmission check pattern, it returns to the main loop and enters the logic for either command.

For the transmit radio check, the program will record everything immediately following the last carrier signal until the carrier signal disappears. Then the program will play back the audio and compute a power level. This power level will be the signal from the ADC that is received and then will be translated audibly to the pilot so they can understand their

signal strength.

For the weather reporting function, the audio recording begins with the current time, then wind direction, wind speed, if there is a gust, temperature, dew point, altimeter, pressure, and density altitude. Many of these values have to be computed from the readings we receive from the weather sensors.

5.3.2 Poll Weather Conditions

The Poll Weather Conditions process is the side process which is started by the main program during its initialization. This process will do all the communicating with the weather sensors and will read and store their values into an object that the main program will request whenever a weather request signal is detected. The program starts by verifying that it can communicate with all the sensors and then it will reset all of its temporary variables. Then it will enter the infinite loop where it polls and stores the readings from each sensor. The temperature/pressure/humidity sensor will only be accessed on a timer because the sample from that sensor has a slight delay and the weather conditions it reads do not change very often.

First the program will read, calculate, and store the wind speed and then it will compare the current wind speed to the last recorded wind speed. If the difference between the two is greater than a designated threshold, the program will label it as a gust. It will only report a gust in the weather object if the difference is detected more than once. To detect it again, we have created a second flag called verifyGust. Once the first gust is detected and the gust flag is set to true, the next time the difference between the current and last readings is greater than the threshold, the program will enter a separate conditional to set the verifyGust flag and report it to the weather object. After it has been reported, both flags will be reset. Next the program will read and store the wind direction. DirCount is a counter that lets us set the period we want to calculate the average wind direction over. Once the counter equals that set value, we calculate the average wind direction using the last set of recorded readings from the sensor and then reset the counter. This average is the wind direction the process will store to the weather object which will be returned to the main program to report to the pilot. If the counter does not equal the set value, then we will increment the counter and continue to the temperature sensor.

Finally, the program will read and store the temperature, pressure and humidity but only when the TempCount counter equals the set value for the designated time interval. This interval will be much larger than the wind direction interval because the values for

temperature, pressure, and humidity will not change very often.

This process will run continually in the background while the main process listens for a carrier signal. When the count clicks function from the main program returns a decision to transmit weather conditions, the program will first call this function to collect the most recent weather value.

There were many possible ways to set up this function for this system but ultimately, we chose this more object-oriented approach because it makes the passing of the weather information between functions easier and creates a structure that allows us to easily store all the weather conditions for a particular period in time. This method also simplifies the code immensely because instead of individualizing each weather measurement, we are able to iterate through all of them with timers to pull new values from the sensors at certain intervals. It was important to use timers for the weather measurements because some of the measurements don't change very often or have a significant delay from the sensor, like temperature, and others require an interval to compute a value or average from, like wind speed or direction.

The following figure is the logic diagram for the weather polling function that visualizes the information described above. It shows the iteration through each of the weather measurements, the check of their corresponding timers, and the resulting pull of new data from the sensors. An important part of the logic in this program is the wind gust detection. It is crucial for pilots to be able to be notified when there are wind gusts because there are certain counter measures they must take in order to keep control of their aircraft and to land safely. The way we have set up the logic for wind gust detection is very accurate and it allows pilots to be confident in the weather condition reading they are receiving from the system. Since we have set two flags that must both be true in order for a gust to be reported, it allows the system to only report when there is a consistent gust instead of a singular event. There is nothing we can do to notify the pilot of a singular gust but it is important for them to know if the winds are particularly choppy near the runway. Another important aspect of the weather polling program is how the wind speed and wind direction are calculated. Wind direction is based off of a potentiometer with a dead zone at 0/360 degrees. This causes some difficulty with the logic since we have to perform an average over a period of time. How do you take an average over a null value? To solve this problem we decided the best way was to detect when winds are varying over the deadzone then find the average and add 180 degrees to find what the adjusted average should be.

The weather polling function itself is also a bit different than when we originally thought it

out. After setting up the weather object with the attribute names, we set each attribute as the result of its corresponding function.

The function to calculate the wind speed stores the last five values from the anemometer through the ADC. The anemometer calculates speed by counting rotations so by using the number of rotations known for 1 mile per hour, we are able to count the number of rotations and calculate what the value would be in knots. Next, we have to take any gusts into account. To do this we compare the current value just calculated from the ADC and the last value that was stored and compute the difference. If the difference is above a set threshold then we verify the gust and set the corresponding flag to true so that when the program audibly reports the weather, it also includes the gust.

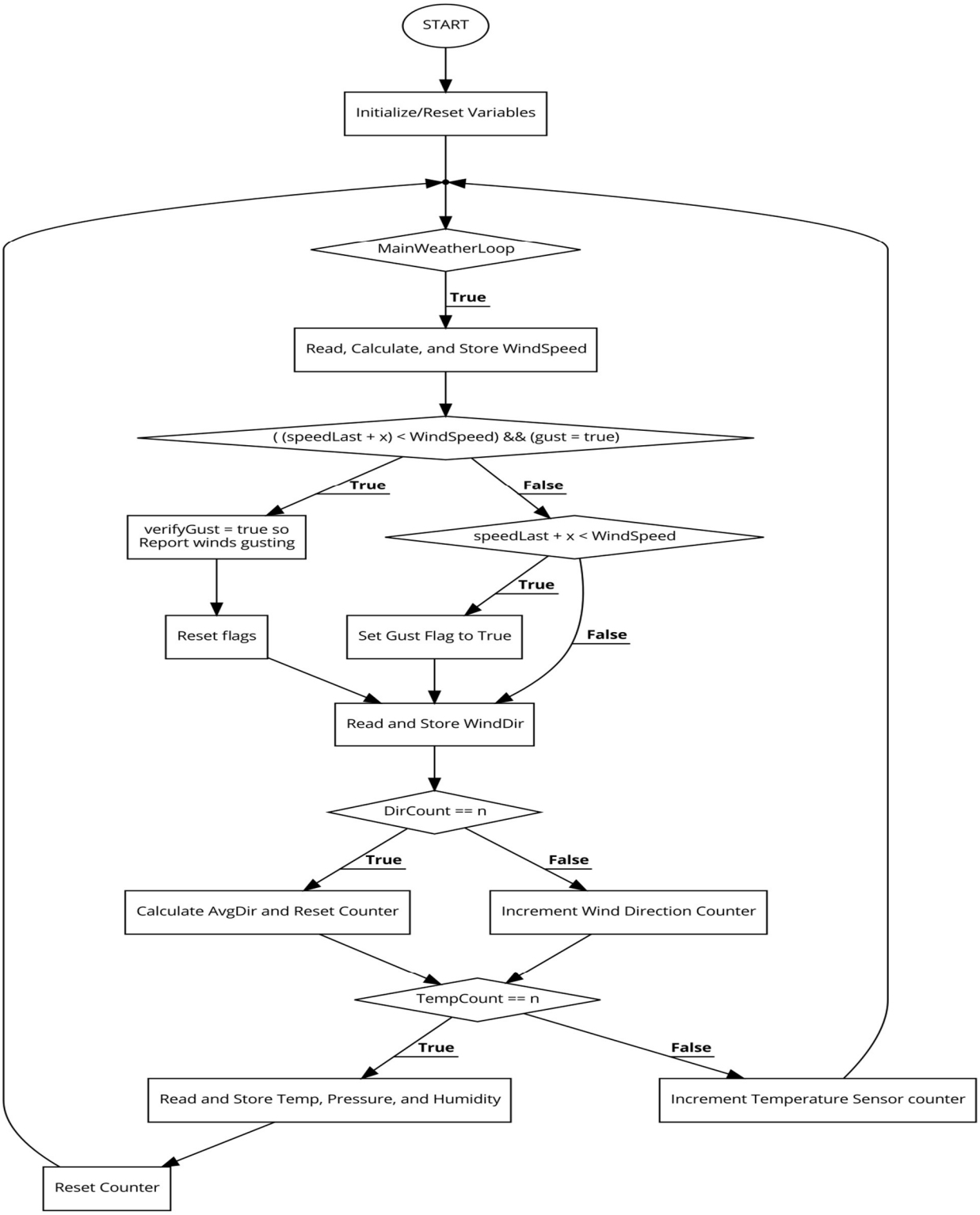


Figure 5.2.2 Poll Weather Conditions

For wind direction, this function became a lot more complicated than anticipated. The issue was with calculating the direction when the potentiometer passes over “0”. Here the value from the ADC drops which keeps you from getting an accurate average if you simply store the values and compute a basic average. To accurately calculate the wind direction, we had to calculate the angle between individual measurements and find the average and then convert to degrees.

Next, we have the function to verify that winds are indeed gusting. This is done by using two Booleans and forcing both to be true before gusts will be included in the weather report.

Lastly, the function to find the current temperature, pressure, and humidity, was exactly as we had originally thought. Since all three measurements are from the same sensor, all the program has to do is call each corresponding address on the ADC and store the value. The translation to the correct units is done before the value gets audibly reported back to the pilot.

5.3.3 Counting Radio Clicks Process

This process counts the number of times the pilot keys their radio and checks the duration of each click or spacing to be sure the program is not picking up accidental clicks or the wrong signals. This process is triggered whenever the main program encounters a click. This process will then time the click, considered the “on” time, and if it falls between specified maximum and minimum parameters, the program will move on to the “dwell” time which is the spacing between clicks. It will continue to check each segment according to the patterns we have designated for a communication check or weather report until either a duration does not fall between the specified parameters or we don’t receive the segment we were expecting. After the second click or “on”, the program will time a “gap” instead of a “dwell” which has a longer duration in order to register a pause between the first sequence of clicks and the second sequence of clicks. If this pause, or any duration, does not fall between the specified parameters, the process will end, ignore the accessed clicks, and will start over to listen for the next new click. Once either pattern sequence is found, the program will decide and escape into the corresponding function to report either a communication check or the weather.

This flow of logic, while not pleasant to walk through, was the most efficient way to access clicks from the pilot. Instead of paying attention to and computing every click, we only

care about the ones that meet our criteria. This way, if there is any click in any sequence that does not meet our criteria, we scratch the sequence and being to listen for the carrier signal again. This reduces some of the background time that would be necessary to access every click and it makes the system more responsive because it only pays attention to the signals it requires.

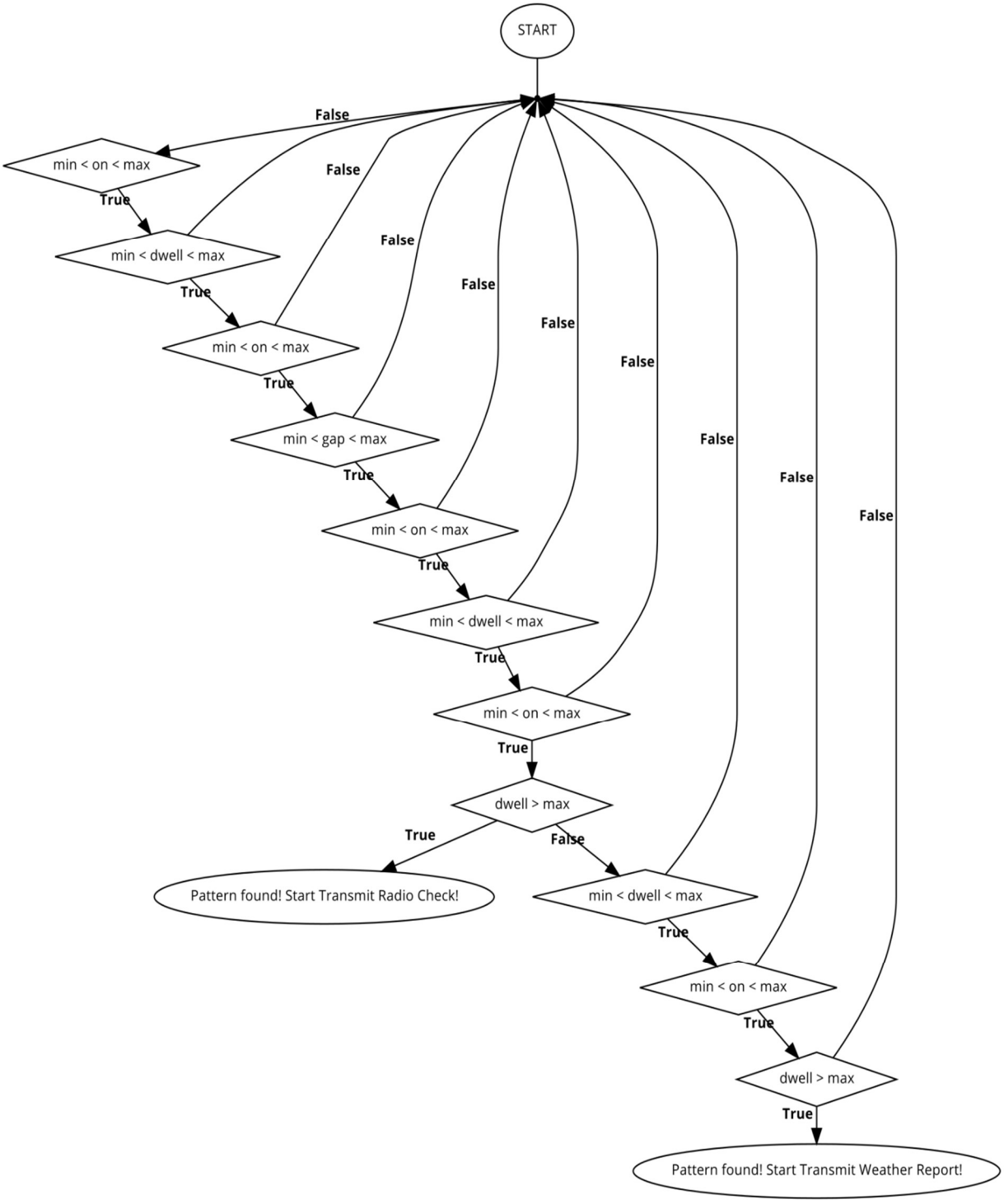


Figure 5.2.3 Count Clicks Process

5.3.4 Transmit Weather Conditions

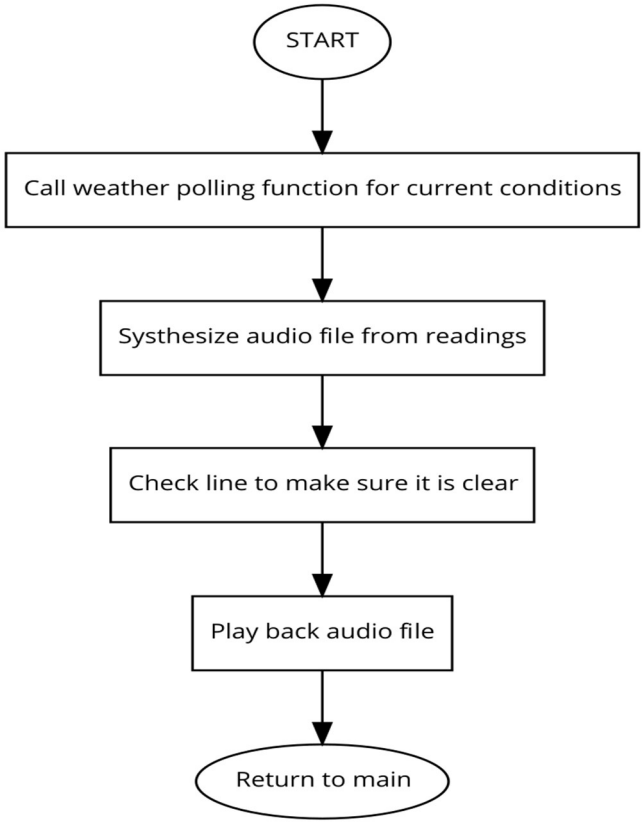


Figure 5.2.4 Transmit Weather Conditions

This process will start after the main function recognizes the click pattern for weather reporting. From there, the main program will make a call to the weather polling process to receive the current weather object. Then the program will separate each piece of the object, collect all the voice files needed to synthesize each condition, and then concatenate them into a single audio file. Once it has created the audio file for the current weather conditions, it will check the transmission line to ensure that the playback does not step on anyone. After it has checked that the line is clear, it will then broadcast the current weather conditions for the airport including wind speed, wind direction, gusts, temperature, pressure, and humidity. This process must be efficient so that there is not a noticeable or substantial delay between the time that the pilot clicks their radio and the time that the weather report starts to transmit.

5.3.5 Radio Communications Check Process

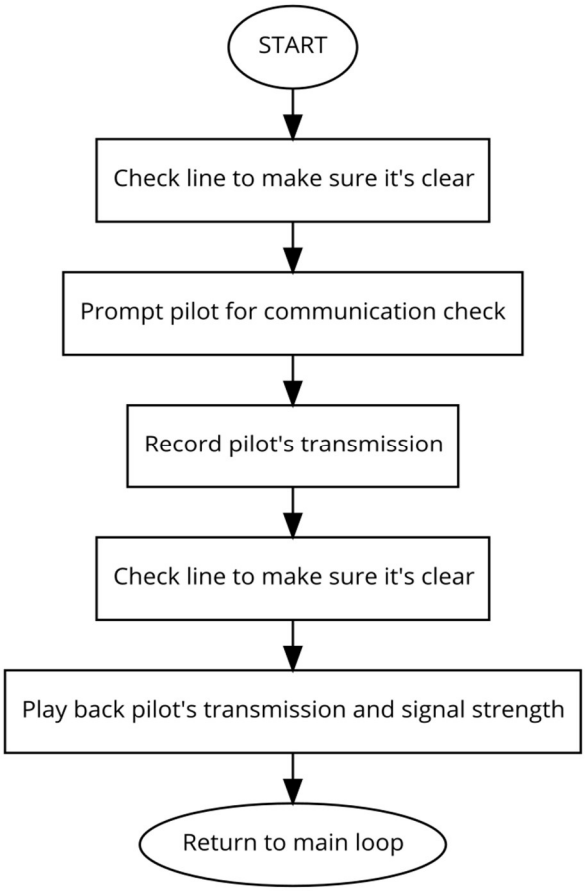


Figure 5.2.5 Radio Communications Check Process

This process will start after the main function recognizes the click pattern as the correct pattern for a communications check. After it has made the decision to proceed with a communications check, the program will check the transmission line to make sure no one else is on the line. Once the line is clear, then it will transmit a prompt to the pilot which will acknowledge their request for a communications check and ask them to proceed with their transmission. As soon as the next carrier is detected, the program will begin recording the audio transmitted and it will stop recording when the carrier is no longer detected. As discussed earlier in this document, we will be using an audio codec which will allow us to efficiently record audio directly into a WAV file to easily play back. This reduces a lot of overhead since we do not have to create the WAV file manually. After the carrier signal is no longer detected, the program will again check to make sure the

line is clear and then it will transmit the recorded audio file back to the pilot. After the audio file the program will also announce a signal strength level based off the recording. This will allow the pilot to get a better idea of the quality of their transmissions and allow them to make adjustments as needed.

5.3.6 Initialization

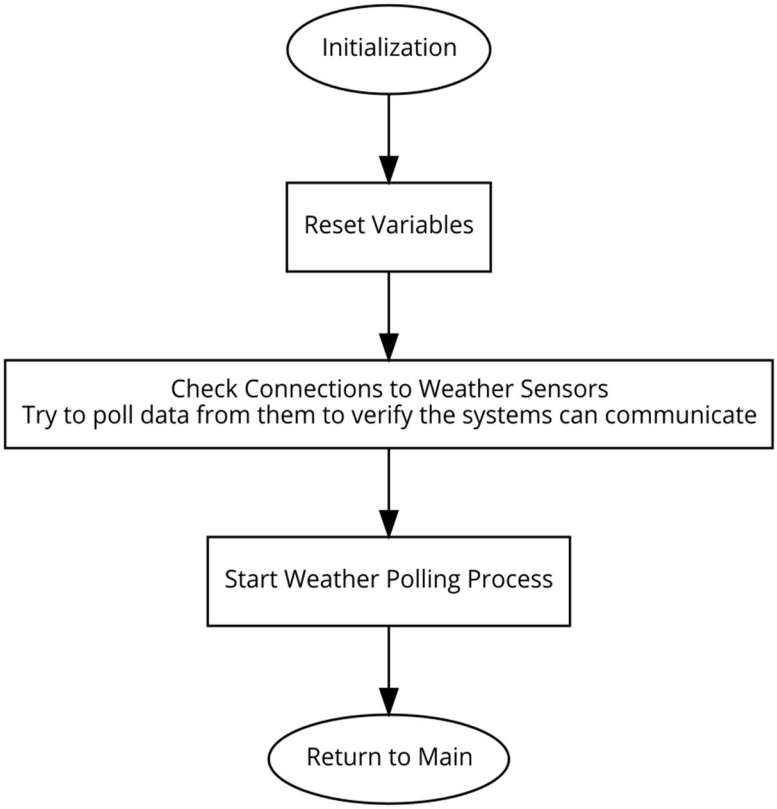


Figure 5.2.6 Initialization

This is the Initialization process of the software. Once the system is powered on, this process will be immediately called and executed. In this process there are three main commands. First, the computer will reset all the variables in the main program. This ensures there are no extraneous values left over from the last time the program was run which could interfere with current readings or calculations and create extraneous results. Next, the software will verify that it can communicate with both weather sensors. Finally, it will initiate the never ending process of polling the weather data, which will gather and record data from the anemometer and temperature, pressure, and humidity sensor and is further explained in the previous sections. Lastly, the software will also start the web

server that will be running from the computer. From there, the software will return back to the Main Loop.

5.4 Communication with Interface Board

5.4.1 Pin Layout

The communication between our interface board, temperature, and weather sensors are directed by our MCU, the Raspberry Pi. The interface board, at its end, interprets all incoming and outgoing signals between the VHF Aircraft radio and the microprocessor. To receive and analyze the analog signals from the radio and weather sensors, the Pi needed to be outfitted with an analog-to-digital converter -we chose the ADS1015 with 12-bit precision. The next step was to verify the best viable way we could connect the ADC to the Pi. The options we researched included either using the SPI bus to connect to Pi to MCP3008 or I2C bus connected to the ADS1015.

Another communication line required for our project is the connection between our system and a web interface. The web interface is one of the ways that allow the user to change the current airport location of the device. It provides the current weather condition to the user using a graphical interface modelled as a compass.

The Raspberry Pi 3 Model B has 40 dedicated pins. The Pi's documentation details each available pin with their respective pin number. The table is also color coded to highlight the specific use of every pin. Of the 40, 26 pins are general purpose input and output pins (GPIO pins) while the rest are ground, power, and two other pins for additional functions. The two other pins are for the I2C Bus that our team utilize to convert the analog data procured from the sensors to digital signal. The rest of the GPIO pins are just used to transmit and receive digital signals. They are used to communicate between the interface board and Raspberry Pi.

5.4.2 SPI or I2C connection

The tables below differentiate the connections required between two possible ADC sources we researched. This includes a connection between a MCP3008 (hardware and software SPI connections) and the Pi and between the ADS1015 and the PI.

MCP3008 (Software SPI)	Raspberry Pi 3
VDD	3.3V (Pin1)
VREF	3.3V (Pin 17)
AGND	GND (any ground pin)
DGND	GND (any ground pin)
CLK	Any GPIO pins (pin 18 for example)
DOUT	Any GPIO pins
DIN	Any GPIO pins
CS/SHDN	Any GPIO pins

MCP3008 (Hardware SPI)	Raspberry Pi 3
VDD	3.3V (Pin1)
VREF	3.3V (Pin 17)
AGND	GND (any ground pin)
DGND	GND (any ground pin)
CLK	SCLK (pin 23)
DOUT	MISO (pin 21)
DIN	MOSI (pin 19)
CS/SHDN	CEO (pin 24)

ADS1015	Raspberry Pi 3
VDD	3.3V (Pin1)
GND	GND (any ground pin)
SCL	SCL (pin 5)
SDA	SDA (pin 3)

The SPI connection (both software and hardware style configurations) requires more physical connections than the I2C bus and creates additional problems when dealing with noise. Problems also arose from the SPI's asynchronous feature as it doesn't guarantee the same clock rate between connected devices. This can cause problems when two system with different clocks attempt to communicate.

The inter-integrated Circuit (I2C) Protocol (also asynchronous) is the route we chose for connecting our Pi to the external analog-to-digital converter. The I2C bus requires less connection (only two lines) and allows us to communicate with multiple devices as illustrated below. The two lines can support up to 1008 slave devices and allows more than one master to communicate with all devices on the bus unlike the SPI connection.

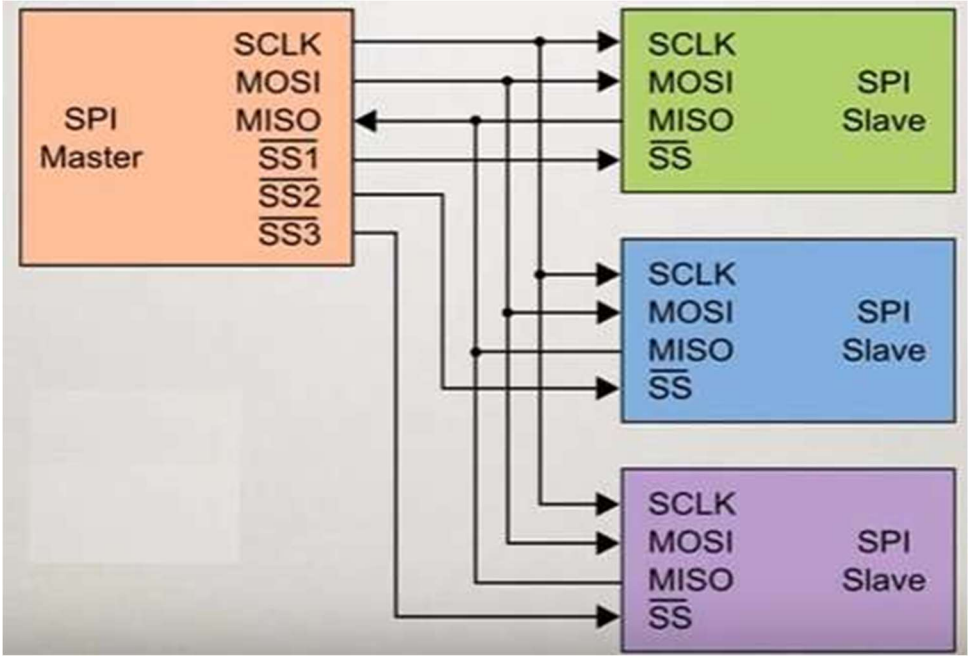


Figure 5.3.2a SPI connected to multiple devices

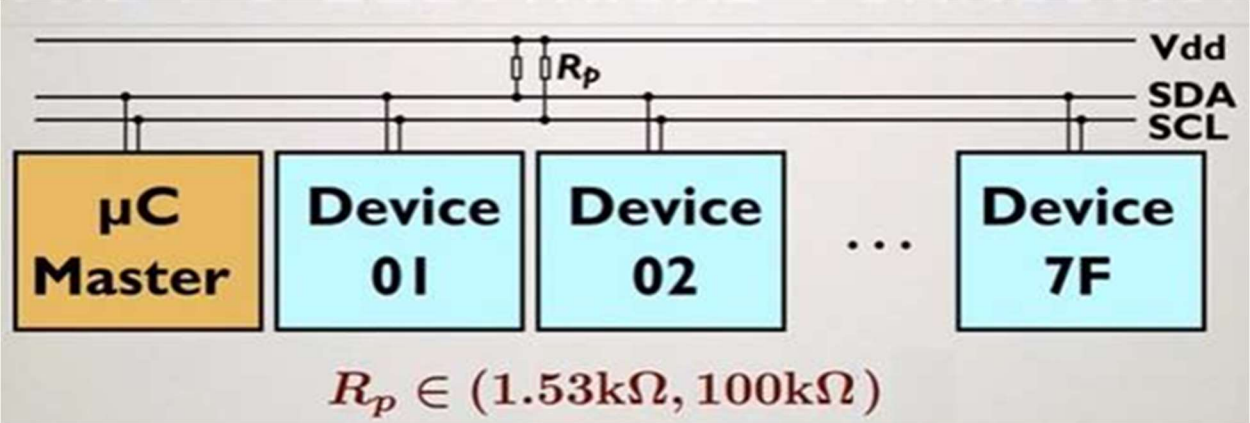


Figure 5.3.2b I2C connected to multiple devices

5.4.3 ADS1015 Communication Logic

5.4.3.1 Background Information

We chose to use the ADS1015 for our analog-to-digital converter. This particular ADC is supported with a variety of software libraries and interfaces that are open-sourced by Adafruit Industries. The open-source libraries and interfaces provided made the overall coding process easier because without these libraries we would have to start coding from scratch. Creating a library would have resulted in a delay in our schedule for we would have to create the functions required to read the analog signals. With the already published libraries, we can skip this step and just call the function required to obtain our data.

Another viable option for an external analog-to-digital converter is the MCP3008. The MCP3008 is also supported by Adafruit Industries through a variety of software libraries and interfaces. We ultimately chose the ADS1015 as our sponsor had mentioned its versatility for obtaining precise analog to digital conversion as well as amplifying and accurately processing extremely low signals.

5.4.3.2 ADS1015 Wiring

As mentioned earlier, the Raspberry Pi doesn't have a built-in onboard analog-to-digital converter like the Arduino Uno. We needed to find a compatible A/D converter with enough power and precision. Our choice was split between two ADCs, the MCP3008 and the ADS1015 -we chose the ADS1015 which uses the I2C bus as opposed to the MCP 3008's SPI bus. The Pi is thus complimented by the ADS1015 external analog-to-digital converter to process and convert analog readings from our sensors to digital signal; the digital signal is then processed by the Pi to obtain and relay necessary information.

The ADS1015 is a 12-bit precision ADC that operates at 3300 samples/second and interfaces via the I^2C communication bus. A 12-bit precision allows for higher accuracy when obtaining, for example, the exact degrees associated with the wind direction. This chip contains 4 single-ended input channels, requires 2V to 5V to run, and includes a programmable gain amplifier that provides up to x16 gain for small signals. The programmable gain amplifier helps magnify and boost smaller signals to be able to read them at higher precision.

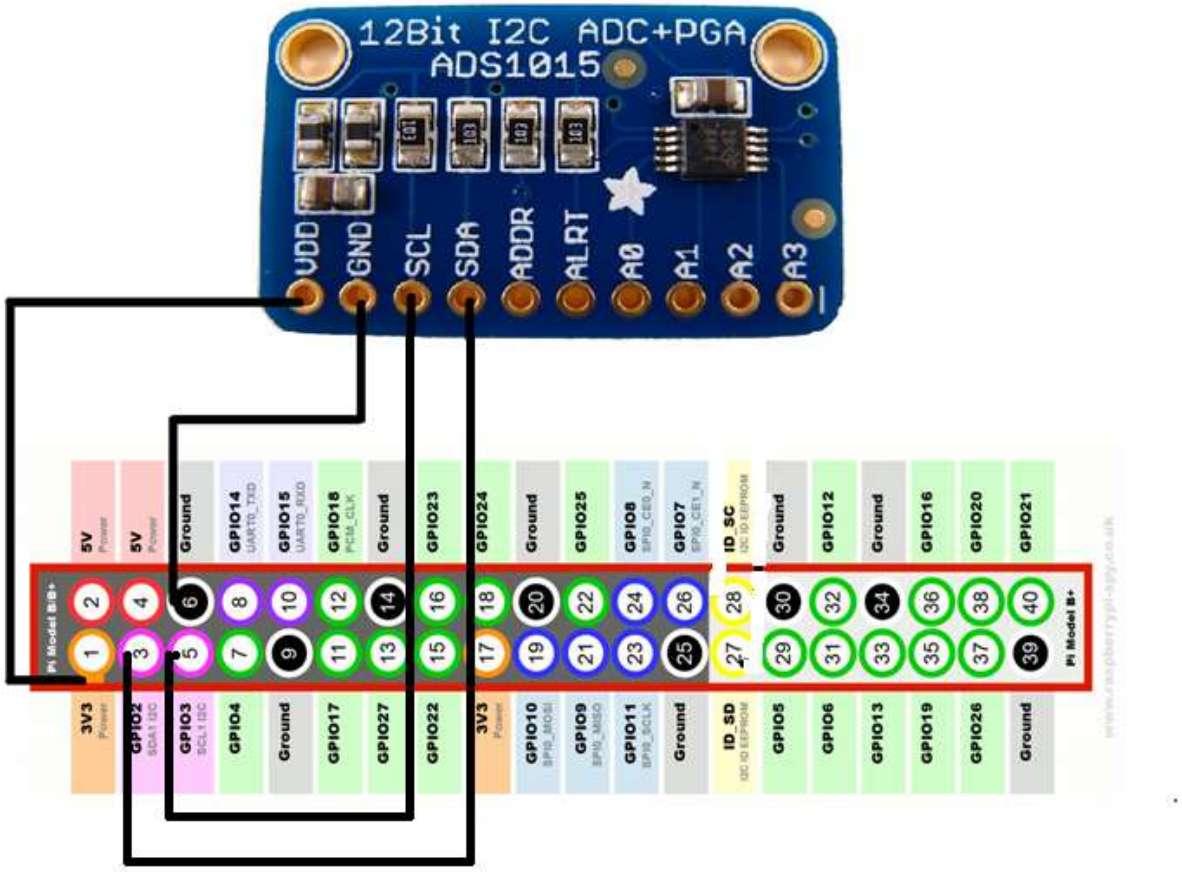


Figure 5.2.3.2 ADS1015 connected to the Raspberry Pi

The wiring between the Raspberry Pi 3 and the ADS1015 is shown above in figure 5.2.3.2. The I2C bus of the ADS1015 makes the wiring fairly simple with no extra step required except on the software side. The ADS1015's VDD is connected to the Pi's 3.3V (pin 1 in our case) as it requires a power source from the range of 2V to 5.5V. The ground pin of the ADS1015 can be connected to any ground pins on the Pi; we connected ours to the sixth GPIO pin on the Pi. The ADS1015's SCL pin receives a clock signal from the

microcontroller and is connected to the I2C SCL dedicated pin on the Pi. The SCL pin is the 5th pin on the Pi. This pin uses the clock signal provided by the microcontroller to clock data from the SDA pin. Data obtained by the sensors is transmitted and received through the SDA pin connected to pin 3 of the Raspberry Pi.

As mentioned before, the ADS1015 supports up to 4 single-ended input channel. This includes input channel A0-A3. Single ended inputs only measure positive voltages but provide twice as many inputs. On the other hand, there are two differential inputs used to measure voltages (with the ability to also measure negative voltages). This analog input is measured between two analog input channels A0 and A1 or A2 and A3. We did not deal with negative voltages plus the increased immunity to electromagnetic noise provided by the differential measurements was ideal for dealing with noise during our testing procedures.

5.4.3.3 Programming the ADS1015

In order for the Pi and ADS1015 to operate properly, we installed Adafruit Industries' required libraries to allow the devices to communicate and ease the code development process. We installed the Adafruit ADS1015 python library. This library allowed us to use several commands like "read_adc_difference()" which reads the voltage difference between channel 0 and 1. The function returns the signal difference between both channels which will allows us to obtain the noise acquired from analog signal inputs from our sensors.

The libraries provided us with many more functions and examples of singled-ended analog to digital conversions as well as differential conversions. These methods allow us to convert analog signals to digital signals as well as setting the gain of the on-board programmable gain amplifier.

5.4.3.4 I2C Interface

Since the Raspberry Pi has dedicated I^2C ports, The Raspberry Pi can communicate with the ADS1015 via the I^2C bus interface instead of its GPIO pins which is much more preferable than a SPI connection (as illustrated in section 5.2.2). The I^2C bus operates between many devices; usually one device operates as the "master" while the others are defined as the "slaves." In our project's case, the master is the Raspberry Pi and the slave is the ADS1015 as well as any other devices connected on the bus. It is important to mention that both master and slave constructs can read and write, but the slave constructs can only do so with explicit permission from the microcontroller -the master.

The I^2C bus operates based on two lines, the SDA and SCL. The SCL provides the clock needed to clock the data received by the SDA line; the SDA line carries data between the two devices. This data is transmitted in chunks of 8-bits on the bidirectional SDA line. When transmitting, the SDA is either high or low, but requires the SCL to be low in order to do so. A high SDA means the bit is 1 while low represents the bit as 0. This receives and transmits data with the terminology that if the master sends file to the slave, then the master drives the data line; else, if the master reads from the slave then the slave drives the data line. The bus lines are idle when there is no communication happening between the Raspberry Pi and the ADS1015. It's worth mentioning that only the master can start the communication between both devices.

For communications to start between the Raspberry Pi and ADS1015, the Pi must initiate the communication to the ADS1015 or any other devices; the Pi then needs to provide an address to detail which slave devices it wants to transmit to. This address is a unique 7-bit address given to each device on the I^2C bus. The unique I^2C addresses are set by the ADDR pin. The ADDR pin allows unique addresses to be selected for each slave device connected to the microcontroller. A great debugging tool and check for potential errors is the acknowledge bit that brings the SDA to a low. The acknowledge bit switches the SDA to low confirming that the data was received.

The I^2C interface provides a great communication line that transmits and receives data between the microcontroller and other peripherals with minimum wiring. It functions primarily on two lines, serial data (SDA) and serial clock (SCL) as mentioned above. One of the reason it is better than SPI for our project is the I^2C protocols that allow any number of masters (microcontrollers) to be connected to any number of slaves (peripheral devices/sensors). The SPI connection requires 3 wires: a SS, SCLK, and a bi-directional MISO/MOSI line as well as one SS line per connected devices. Using the I^2C bus, we can communicate to any of the sensors and other devices by using the 7-bits unique slave address assigned to each device with only two lines.

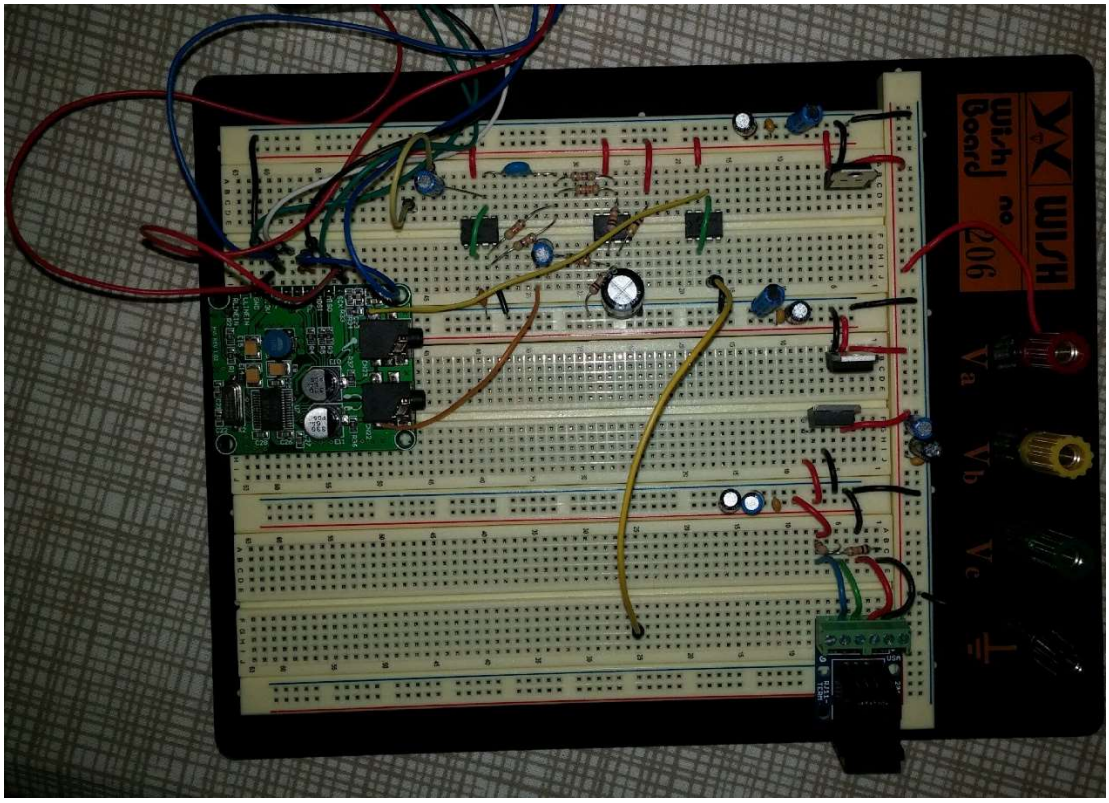
5.5 Configuration Screen

When the user connects to the Raspberry Pi's Wi-Fi hotspot, the user will be able to access the website hosted on the Pi. The main screen (Fig. 6.X on left) will display an overlay of the runway at the airport with a compass rose and an arrow telling the user what the current wind direction is. It will also display the current wind conditions in words below as they would be broadcast to pilots. Towards the bottom of the screen the user

will see three links. One is titled “Archived Wind Data” and when clicked, will take the user to a screen that shows past logged wind data for a specified length of time. Another link is titled “Archived TX Checks” and when clicked, will take the user to a screen that shows past logged TX check recordings for a specified length of time. The last is titled “Change Parameters” and when clicked, will take the user to a screen (Fig. 6.X on right) where they can change every aspect of the system including, but not limited to, runway headings, carrier dwell time, and the number of clicks for functions.

5.6 Integration and Prototype

This section describes how the components are integrated and the breadboarding that has been done to combine the components.



Here we have the RJ11 network hooked up to be able to measure both the pulses for the wind speed as well as a voltage potential from the wind direction potentiometer.

5.7 Web Server

The web interface is intended to provide an easily accessible graphical interface for the user. The interface would provide the user with valuable information concerning the current weather conditions; this includes wind speed, wind direction, gust, temperature, pressure, and humidity. The interface would allow users to check the current conditions anywhere at any time. The system will also allow the admin user for that airport to switch the click pattern for requesting different tasks, like a communications check, to best fit their preference; the administrator would also need to switch the click pattern if the current click pattern interferes with any patterns already established at a specific airport. For example, those conflicting patterns could be the queue to turn on the runway lights or for other airport announcements or communications.

5.7.1 Introduction to the Model View Controller Architecture

Since we did not want to deal with the arduous process of creating and implementing a big and complicated relational database we looked towards other more simple and practical options. We decided that a web framework based on a Model View Controller architecture would best meet our project's needs. This type of web design is simplistic and allows us to easily transfer data between the frontend of the system to its backend. Focus would be set at the back end of the system meant for capturing the weather data, passing, and formatting it into a relevant and easy to use database. We would then use the data obtained from the backend and broadcast it back to the frontend without having to deal with any of the complicated PHP scripts like PHP or MySQL. We would not need to use any PHP scripts with a Model View Controller framework in order to send queries back and forth to the database. This would further decrease the complexity of the development process. An increase in performance would also be achieved because if we're using a Model View Controller architectural pattern, the system would not need to load the page and recommunicate to the backend for the specified data every single time. Thus, the increase in performance since that's one function we do not have to repeat over-and-over again.

This kind of architectural pattern further increases the performance of our system since the data received is dynamically allocated to the class-based views structure. A view is a

callable that obtains a request and returns the appropriate response. The class-based-views structure allows us to rapidly structure our views dynamically; they are saved and can then be accessed and reused through inheritance and mixins. This is also an alternative way of implementing views as Python objects instead of functions or methods. The view class handles linking the view into the URLS, HTTP method dispatching, and several other simple features like redirectView and templateView. This provides multiple benefits as the codes related to any specific HTTP method can be utilized by separate methods; not just through conditional branching access. This also increases ease of use for our application because we can use multiple inheritance to pass down the object and reuse its components. The Model View Controller increases performance, allows for more efficient code reuse, and parallel development by decoupling its major components and focusing on each separately and simultaneously.

As mentioned earlier, the team did not want to deal with the complicated structure and code development associated with designing and implementing a needlessly big relational database. As a relational database would not only be impractical but would also cause major performance issues when operating with the Raspberry Pi. Instead we decided to use a web framework based on the use of a Model View Controller architectural pattern. This design model would allow us to parse and format the information obtained via views and not through the use of complicated PHP scripts; in other words, this would make obsolete the need to request and send multiple queries to the database every single time data is required. In the Model View Controller architecture, the controller component steers the entire system. The controller does this by handling all requests and responses across the database. It sets up the database connection and handles loading addons. It obtains and reads a setting file that feeds it the info regarding what to load and set up. Furthermore, the controller component is provided an URL configuration file that instructs it on the desired responses from an incoming request from the browser. On the other hand, the model partition of the architecture captures the required data the website needs and stores it into database tables. Fortunately, Python provides extensive examples detailing exactly how this is done. Python classes (or models) are emphasized and work quite well with the Django framework that tie into a one-to-one ratio the database tables. Switching to another component, the view is the user interface layer. It provides an automatic web admin interface for editing the models

using the python code.

This type of design steers and controls data more efficiently with less load capacity than a regional database which is great and definitely meets our project's requirement. A regional database would just be too complex and would result in a decrease in performance as it would request and acquire the desired data then parse that data to and from the frontend and backend of the system repeatedly. The propose MVC architecture is faster as we obtain the desired data and simply pass it using the model. Then we can pass it to our views and allocate the database dynamically without any complicated implementations of PHP scripts.

5.7.2 Django Web Framework

We researched a few Model View Controller frameworks and found Django. The Django Web Framework is quite a robust and great selection for the backend of the system. Django provides a fully functional backend web frame work with the admin view application. It provides a concise and picture-perfect style with multiple features; unfortunately, it does not provide a good template for the frontend application. We then realized we could apply a different framework for the frontend and proceeded to look for a compatible version which will be discussed in greater details later in the next subsection. We chose the AngularJS for the frontend of the system, the parts visible to the user like HTML, CSS, client-side JavaScript, because we didn't want to deal with creating our own template from scratch and wanted to avoid html coding. Since time was of the essence, we looked for an already customized and optimized frontend template. This made the frontend development quicker and allowed us to spend more time working on the Django backend and customizing the views and database model. We researched multiple frameworks as well as a few platforms that would support our project and best fit our capabilities. We looked over AngularJS, JQuery, and ReactJS as viable options to see which would work better with Django. We finally decided to use the AngularJS for the front-end framework of our system.

With the Django MVC style framework, we are able to further simplify the coding development process because we can write our code in an object-oriented manner and use the framework to build our database table simultaneously in the background; this

framework abstracts a lot of the database behind models that are represented as python objects. Each table database can be treated as an easily accessible object making it quite useful because we are able to change the fields of that record with no big fuss. Its fields are treated as general variables that are part of the object. Since the database is abstracted, we can import the schema of our models directly into our views. We can then basically treat the database records as if they were objects and insert them directly into our html code. Another major reason that this method is useful, is the fact that we do not have to write anything in SQL. We are able to use functions to obtain and apply the specified fields of that object of our database in order to store, sort, and search through the database; we can then sift through the data, update, and record the database without having to worry about coding in SQL. This framework is definitely great for our project's purpose and meets its designed specifications; as mentioned previously, we can choose any database we want for the backend and not have to change or worry about compatibility issues with our models and any related issues to the frontend from using a different framework for the backend. This web framework provides extra usability which relieves the coding process as the user does not have to follow the complex steps when dealing with HTML coding. Usually, we would have to create a client, discern the correct SQL statement, and recommunicate to the backend system. Then it must wait for the response to our request. Instead, this framework relieves and negates these steps as it is more flexible and compound the user with the ability to use the frameworks custom tags to preload the required data and make use of that database's objects directly in the HTML. Thus, this allows even more flexibility as we are able to change our database based on our needs at any given time; as an example, if we decide we want a smaller, faster, lighter, or more robust system.

The frameworks custom tags and filters, mentioned above, reduce the amount of coding in HTML by allowing the user to utilize prebuilt functionalities in Django. Those functions are designed to address the presentation logic needs of a variety of applications. The custom filters are part of the python functions and take in one or two arguments unlike the custom tags that require a number of arguments to return the correct result. These template tags provide great usability. They are useful because we don't have to write multiple blocks of the same HTML code repeatedly. It also allows us to reduce delays when processing the data received from the database by not having to continuously

reassign data to each block individually. Django also provides many other functions, packages, and modules to further cope with the code development process for the backend. Some of the functions have already been explained above while the modules available for the Django framework like Django Rest and Celery are discussed below.

5.7.2.1 Django Rest

Django's prevalent modules and packages include a variety of API creation framework and other asset managers. Among these API creation toolkits, which are all reusable, is the Django Rest, Django TastyPie, Piston, Django-Nap, and many others. Below is a table providing a comparison between the listed toolkits.

	Rest	**TastyPie**	**Piston**	**Django-Nap**
Applications	202	88	69	1
Development /status	Production /stable	Beta	Alpha	unknown
Documentation	Yes	Yes	N/A	N/A
API key authentication	Yes	Yes	No	No
Serialization	JSON JSONP HTML …	JSON JSONP HTML …	JSON Django	JSON
Accept Headers	Yes	Yes	No	No
Browsable Web APIS	Yes	No	No	N/A

Figure 5.6.2.1 Table of different API creation toolkits

After researching the different available API creation toolkits, we decided to use the Django Rest framework for several reasons. This framework's toolkit, as opposed to the others listed in the table above, definitely has more support and flexibility than the other API's. It is supported by over 202 applications. It is also the most stable and is still in production providing several continuous updates. The ability to code using this toolkit is further increase for beginners because of the multitude of documentations available;

making it easier to learn, understand, and develop. Last but definitely not least, it provides a web browsable API which further helps us with our development process paired with the provided documentation. A web browsable API is a generated API that includes an HTML version that allows for browsing and editing the API. The Django Rest framework is a powerful, sophisticated, and flexible toolkit for building web APIs. It requires both python and Django to function properly and provides support with a variety of packages. We used the coreapi package for schema generation, the Django-filter for filtering support, and Markdown to support the browsable API. We decided to pair this toolkit with Django because of its easy to use and attractive web browsable version of the Django API. Another major reason is the option of returning a raw JSON. JSON (JavaScript Object Notation) is an easy to use lightweight data exchanger that works between a browser and a server where the data can only be text. It allows us to convert any JavaScript object into JSON and send JSON to a server.

The Django Rest framework provides a flexible and powerful model serialization and displays data using standard function based views. With the built-in model serialization for data formatting, we are able to compose powerful representations of our data that is processed and delivered in a number of formats with a few lines of code. Rest is defined as “Representational State Transfer” and allows us to take advantage of Django’s ability to abstract away the database as objects; it also allows us to communicate data to the frontend framework using web endpoints. As previously stated, we are able to provide information to the frontend as a raw JSON which are objects that are used in JavaScript as if we obtained it directly from Django. This is important and worth mentioning because it allows us to parse our data to the frontend framework while putting less stress on the frontend framework. The purpose of using two different framework is because this process allows us to relieve stress on our Raspberry Pi allowing for faster performance and not crashing when obtaining a great multitude of web requests. The simple fact that the Django Framework does not provide a pre-built frontend template also affected our decision to choose a different framework for the frontend portion. A prebuilt frontend template would lessen the coding development process making it easier on the team saving time and also removing the need to write the template from scratch. Fortunately, AngularJS extensively meets the desired requirement for a pre-built frontend template.

5.7.2.2 Celery

Celery is a powerful, production ready asynchronous job queue that allows the user the ability to run multiple python applications in the background. This would allow us to asynchronously queue, schedule, and run functions written as tasks. This system meshes perfectly with our other frameworks as it powers these applications and quickly responds to user's requests. It creates the asynchronous job queue and passes long running tasks to the queue. We installed this asynchronous task queue for Django using Celery and Redis. For this job queue application, it is worth mentioning that we were provided with two major possibilities that would pair quite perfectly with the Django Celery Module. These two major options are Redis and RabbitMQ. Both options are compatible with Celery and are both default recommendations by Celery's developers. RabbitMQ is a fast, lightweight, and persistent job queue that exchanges data between processes, applications, and servers. In this case between Celery, servers (Django), and possibly other applications. It is a message broker and message brokers act as a middleman for various applications and reduce loads and delivery time of web application servers. Since tasks usually take a while to process, RabbitMQ or Redis can speed up this process as it is the only job they are meant to perform -so it's best to perform it extremely well.

As previously stated, RabbitMQ is actually faster and a more lightweight and persistent job queue than Redis. But Redis is more robust and can serve as a key-value pair dictionary that is stored in the system's persistent memory. Redis also boasts the potential for having multiple job queues clustered together as to increase performance. The key-value pair dictionary would benefit us in case we found compatibility issues. Furthermore, this process would help us because it would be better to avoid implementing the python objects as read and write for the GPIO pins in order to govern the Django framework and web server completely. The Redis Key Value Dictionary would allow us to store all the different signals obtained from the external peripherals as the Redis key value that would act as a standalone function. This process would provide much more functionality and ease the testing and debugging process. The RabbitMQ software, as mentioned earlier, is indeed faster, lighter, and more persistent than Redis; but, Redis makes up for this shortcoming by providing more functionality that will help us link all the various components attached to our system quite neatly.

Redis, as part of Celery, is used to broker messages between Celery and other applications. Celery is a great choice for our system as it relieves stress on the Raspberry Pi. As mentioned previously, Celery relieves pressure on the Raspberry Pi leaving the software applications created to calculate, process, and output the parameter to take most of the computation power.

5.7.3 AngularJS Framework

The AngularJS Framework is another Model View Controller that we decided to pair with the Django Framework. The Django framework serves as the backend of our system providing a multitude of functions and is quite robust. It is written in python and provides framework custom tags to preload the required data and injects that databases objects directly into the HTML. Django is facilitated by Django Rest as well as Celery.

The MVC framework of the AngularJS provides us the exact requirement we need for our frontend with a focus on developing a single web page application. AngularJS is a structural framework for dynamic web applications that lets the user use HTML templates. It also provides an ease of use for the user by allowing the system to extend the HMTL's syntax detailing the application components clearly and concisely. This is an ideal mesh with our Django framework working with the backend since all the bindings and dependency injection eliminate much of the coding process. This framework is compatible with most current server technology as the data binding and injection all occur within the browser itself. AngularJS offers a better and much simpler format for designing application and is fairly beginner friendly; as opposed to HTML's complex and difficult coding process. AngularJS uses JavaScript in order to teach the browser new syntax after creating new HTML constructs -these constructs are called directives. With these directives, users can break up a single page and separate it into multiple views. This is done by obtaining data from our Django Rest API and storing the data as models. Utilizing both the models and views acquired, the framework easily displays the information requested by the user onto the screen.

AngularJS simplifies applications development process by creating a higher level of abstraction. Using this method allows its user a much more needed control because we

can decouple the client side of an application from the server side allowing the development of both sides to operate simultaneously in parallel to each other. It also allows the system to be able to reuse both sides as needed. This in correlation to the models working in conjunction with the views are all loaded at once. The system only requests and pulls the information it requested as it needs it -and thus the data is dynamically allocated like any Model View Controller architectural framework. This is superb as it allows the AngularJS frontend the capability to obtain all the views from Django (the backend) and combine them into one view. Then the browser can handle the requests for exactly which view and information is currently being demanded. AngularJS truly eases the development process by broadcasting any application easily using services that are auto-injected into a chosen application. This would allow us to quickly create and control the initialization of automated tests.

Utilizing both the Django framework (as the backend) and the AngularJS (as the frontend), all the desired views are all already loaded at once on the client machine. Thus, the system does not need to keep requesting a new view from the Raspberry Pi, recommunicating to it and waiting to acquire its response. The browser is then responsible for switching between the contents that it wants to display and those that it wants to hide. This reduces considerably the traffic and computational load that the Raspberry Pi would have initially observed. Instead, all the computational intensity is transferred onto the client's browser -which is perfectly fine. This works because the client is usually placed on machines more suited for heavy and complex computations. We have discussed the individual frontend and backend framework with the respective packages we intend to use but still have not exactly explained which database engine we intend to use. In the prevalent and fervent spirit of decreasing or completely removing the computational load done by the Raspberry Pi, we used the Django framework. This framework relieves the computational complexity of our system by abstracting away most of the database. The database chosen is still a crucial factor and should definitely focus on being light and fast to further fit our application creation theme. We fixated on the SQL databases like PostgreSQL which provides an extensive number of unnecessary features that our project does not require. SQLite on the other hand, just as the name suggests, is a lighter SQL database that focuses more on speed, memory load, and portability.

5.7.4 SQLite Database

SQLite is a software library stored in a single file format that favors a light and faster database engine as opposed to the heavy traditional database design. It is the most widely used SQL database in the world. It's software libraries implement a self-contained and server-less transactional SQL database engine that facilitates incredible portability. This is made apparent by its simple back up procedure that stores and saves files at certain stages providing the system administrator with a variety of functionality. It provides the administrator with the ability to back up and roll back the database in case it gets compromised or corrupted. This database engine allows files to easily be copied and transferred to completely different systems (as long that it is configured properly). This provides the new system with a complete copy of the database. As mentioned, the database's design is incredibly light which means it takes less space than the other traditional databases. Its small size offers many great advantages that for example allow it to be paired particularly well with the Raspberry Pi. It also boasts of less memory consumption, a great variety of application, and we almost forgot to mention that it is completely free for use for any purposes -private or commercial.

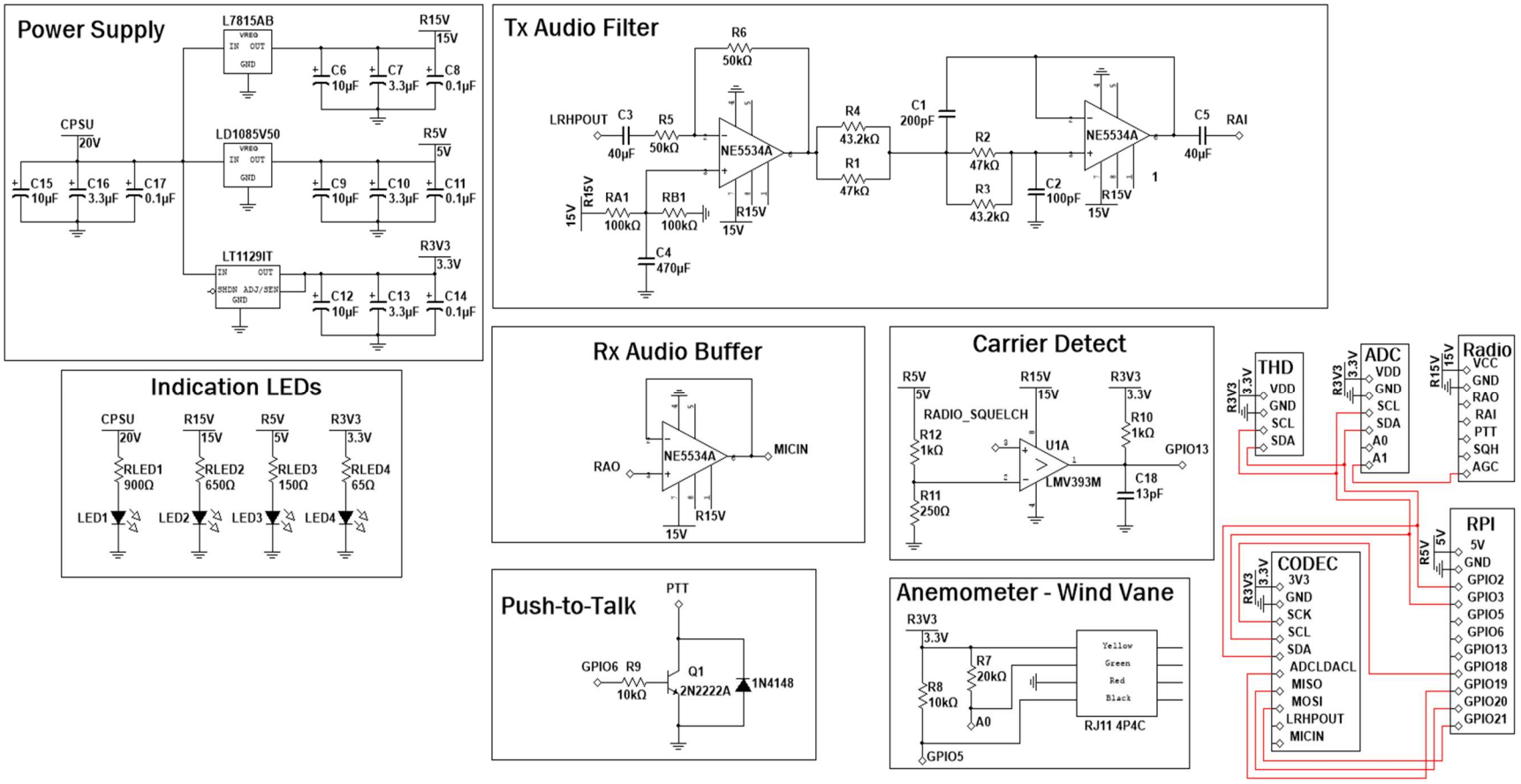
SQLite is a compact library with less than 500KB space necessary to encompass all of its related features. It is a zero-configuration database which means that it does not need to be installed in order to be utilized -no server processes need to be configured. "The system just works" as described by its developers; if the system happens to crash, nothing needs to be done it will re-orientate itself into working order. The databases small size propagates its speed allowing it to work at a very fast pace. Now these advantages we listed are great and pair nicely with our system and the few disadvantages are barely worth mentioning. One disadvantage of this system is the (basically) zero security features it provides. As described above, this is not necessarily a drawback for our system as no personal information is ever saved or even recorded by the system. The system's main priority is to obtain the different signals from our sensors and simply output the correct weather conditions or establish a communications check.

Another possible issue that might arise with other systems is the fact that the database can only allow one write action to happen at one given time. This is not a notable issue

for our system because our system only writes to the database for certain circumstances in which data is being accessed and read from the various peripheral devices connected to our system. After the analog signals (switched to digital) are received, each request sent based on the information obtained can only be processed one at a time. In other words, we do not need to worry about problems arising from obtaining and writing multiple entries at the same time. Worst case scenario a queue system would need to be implemented. Furthermore, this problem would only be an issue if the system was being accessed by multiple users attempting to change the configuration settings over via the user interface at the same time. A quick solution to this is to only allow one user at a time to access the interface at any given point. The simple fact that the Raspberry Pi will only be placed on a local and very small network means that it would be inaccessible to the outside world further trumps the idea that security would be a possible problem. In fact, the only form of security employed by our system is the WPA encryption of the local network the system will be connected to. A would-be assailant would need to have remote access to the Raspberry Pi's DCHP server to access the page and gain the ability to change the configuration settings. Or else, they would need direct access to the Pi.

Another benefit of SQLite is that it is the default database model included with Django. Thus, with any Django installation, SQLite is the type of database that is automatically implemented with the model's page. This is very beneficial to us because of it's simplistic nature, added security benefits, and ease of use. Since it comes bundled with Django, all we have to do is create the Django model based off of our Python weather object and it will automatically create the database which we can then populate with each weather reading.

5.8 Master Schematic



6. Testing

This section describes how the system and each component will be tested to ensure accuracy and to make sure each requirement is met including sub systems. Testing is a crucial part of a system's development because it is necessary to make sure each requirement for the system is met and that the system operates as expected and is reliable. Especially with a system like ours that pilots will be depending on so that they can take off and land safely, it is important that the system be reliable and accurate.

6.1 Anemometer and Wind Vane Testing

The testing of the anemometer and wind vane interface is easily done by measuring the voltage level and waveform pulses at the output of its RJ11 jack. Facing the fin of the wind vane as depicted in figure 6.1.1 should show a reading of 3.3 V on the green output pin, this is the North configuration of the wind vane. The East, South, and West configurations should result in 1.8 V, 1.97 V, and 2.6 V, respectively on the green output pin of the wind vane that is connected to the ADC. These voltage levels were measured from the prototype. The anemometer is tested by spinning the cups and measuring the pulse waveform from the black output pin. This pulse is active low and should have a width of 4.55 ms, shown in figure 6.1.2 below.

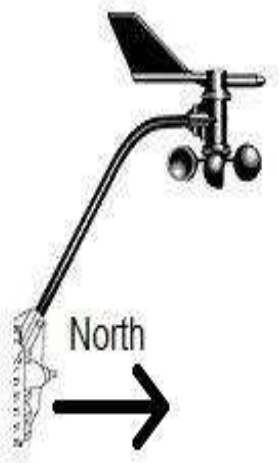


Figure 6.1.1: Wind Vane North Configuration

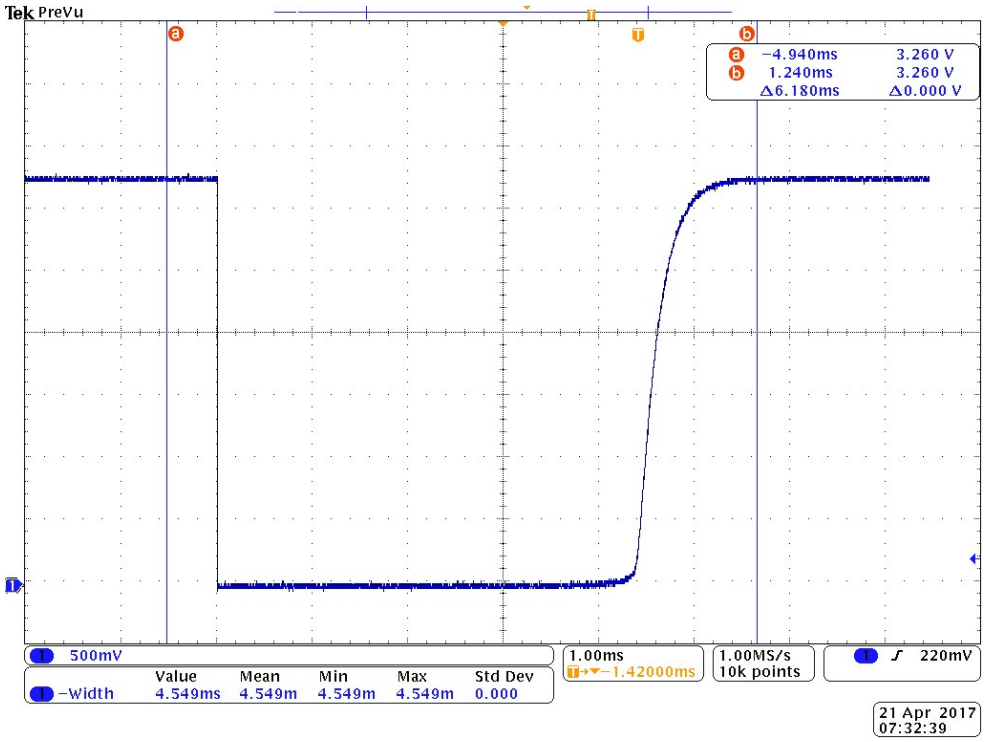


Figure 6.1.2: Anemometer Pulse

6.2 PTT Testing Procedures

Process	Expected Outcome
Turn off the Raspberry Pi 3 microcomputer.	All signals going to the radio should be silent and the Raspberry Pi 3 microcomputer should be off.
Measure the voltage going into the KX 170B Aircraft Radio pin 40	The voltage going in should be 0V
Press the button on the interface board labeled “PTT Test”	The LED labeled “PTT” should light up when the button is pressed
Measure the voltage going into the KX 170B Aircraft Radio pin 40	The voltage going in should be ##V (still need to test for actual value). There should be an audible ‘click’ as the PTT voltage in the radio gets pulled to ground.
Measure the voltage running through the resistor to the Raspberry Pi 3 GPIO Pin	It should be no larger than 1.2V

To test the Interface Board PTT Circuit, follow the testing procedure above.

6.3 Audio Testing

The audio buffer from the radio output to the input microphone of the CODEC can be tested by applying a sinusoidal wave no more than 20 kHz and 3 V peak-to-peak at the radio output (non-inverting input of operational amplifier) and measuring the output of the operational amplifier. Both signals should be identical. This buffer was tested on the prototype by a 3 kHz 2 V peak-to-peak sinusoidal signal, shown below.

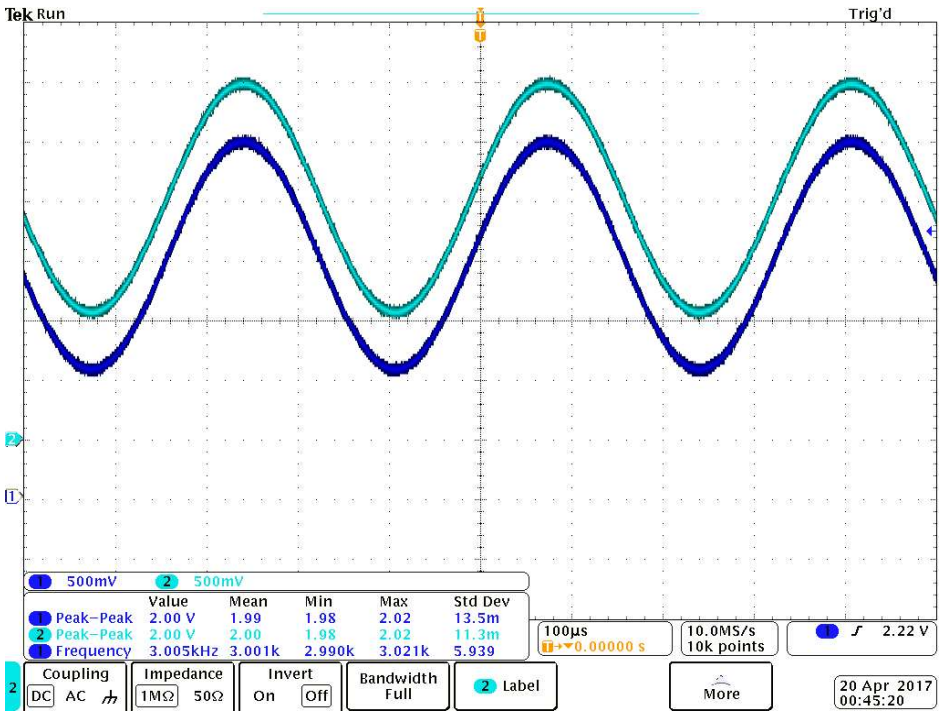


Figure 6.3.1: Audio Buffer Testing

The Butterworth filter from the CODEC audio output to the radio audio input can be tested by applying sinusoidal waves in a frequency sweep at no more than 3 V. The input should be applied at the CODEC audio output with its bias before the DC block capacitor. The output voltage should be measured at the radio audio input after the DC block capacitor. The passband of the filter should result in little to no attenuation in the output voltage. At 50 kHz the gain of the filter should be -3 dB. From frequencies ranging from DC to around 10 kHz there should be negligible attenuation with 0 dB gain. Shown in that table below are the measurements from the prototype filter accompanied with a few waveforms from the measurements. This prototype was implemented with resistor values of 47 kΩ and 43 kΩ and capacitors measured to be 137 pF and 180pF, as they were provided by the on-

campus lab. Due to this, the measurements are slightly offset compared to the designed filter, however its functionality is still sound even with component values different than the design for its function.

Frequency (Hz)	Gain (dB)
100	0
500	0
1000	0
5000	0
10000	-0.22
15000	-0.35
20000	-0.63
30000	-1.26
40000	-3.22
50000	-5.19
100000	-17.02

Table 6.3.1 Prototype Filter Response

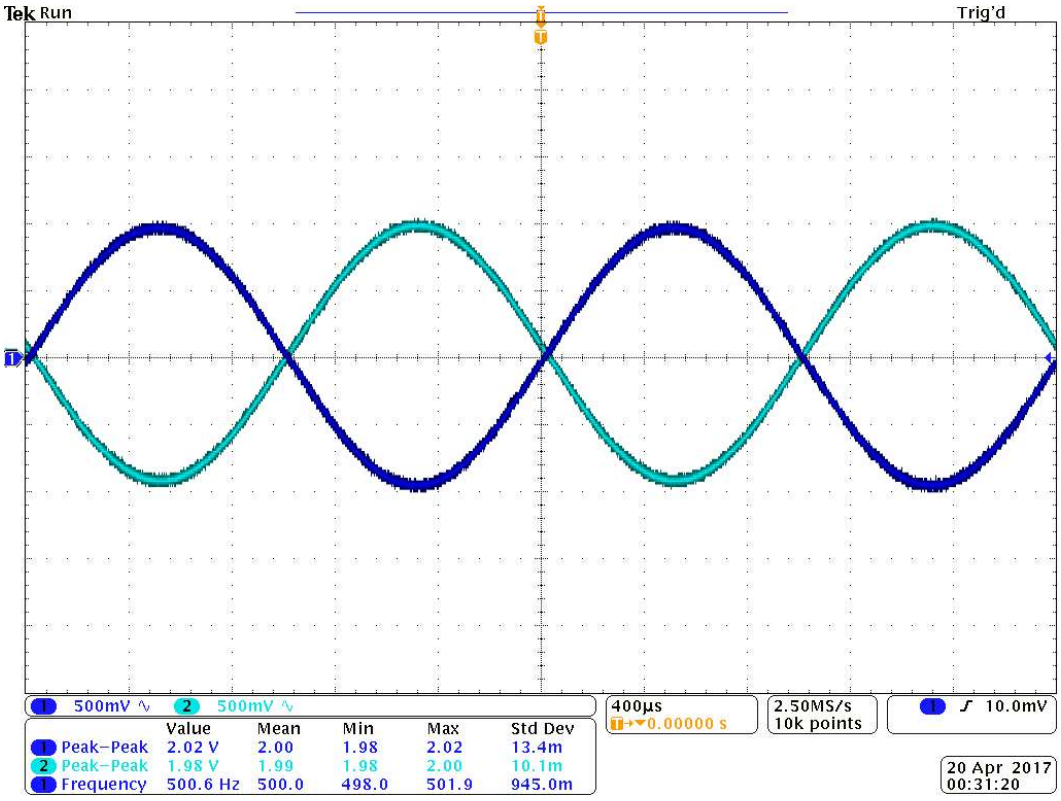


Figure 6.3.2: Prototype Filter Response at 500 Hz

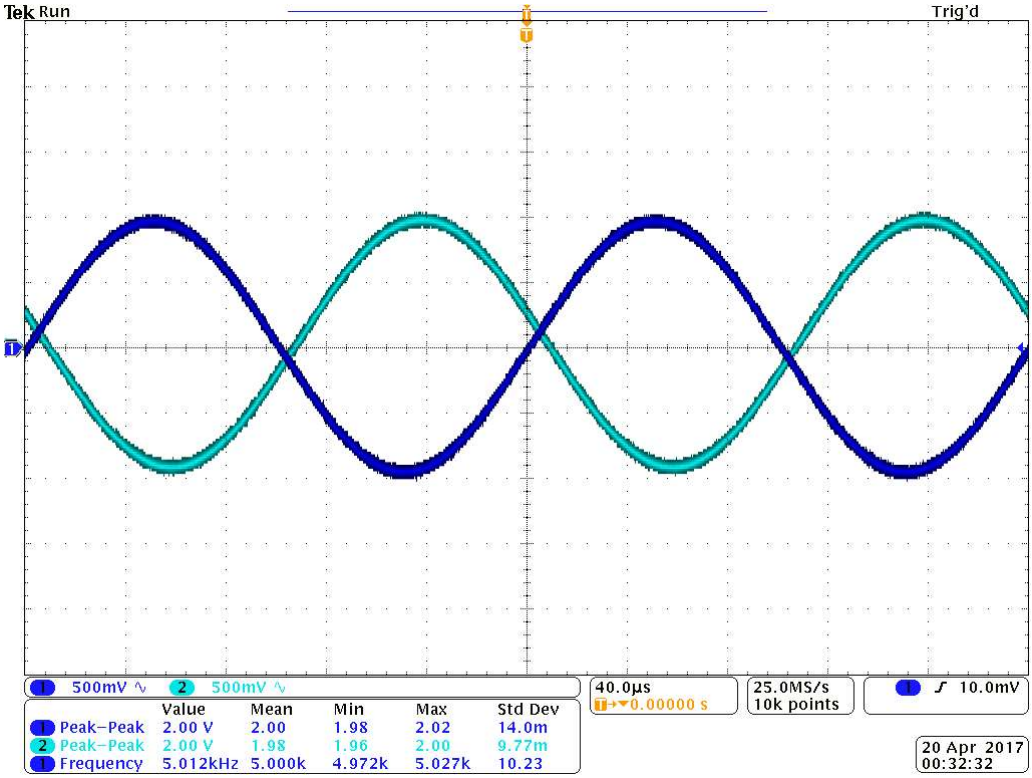


Figure 6.3.3: Prototype Filter Response at 5 kHz

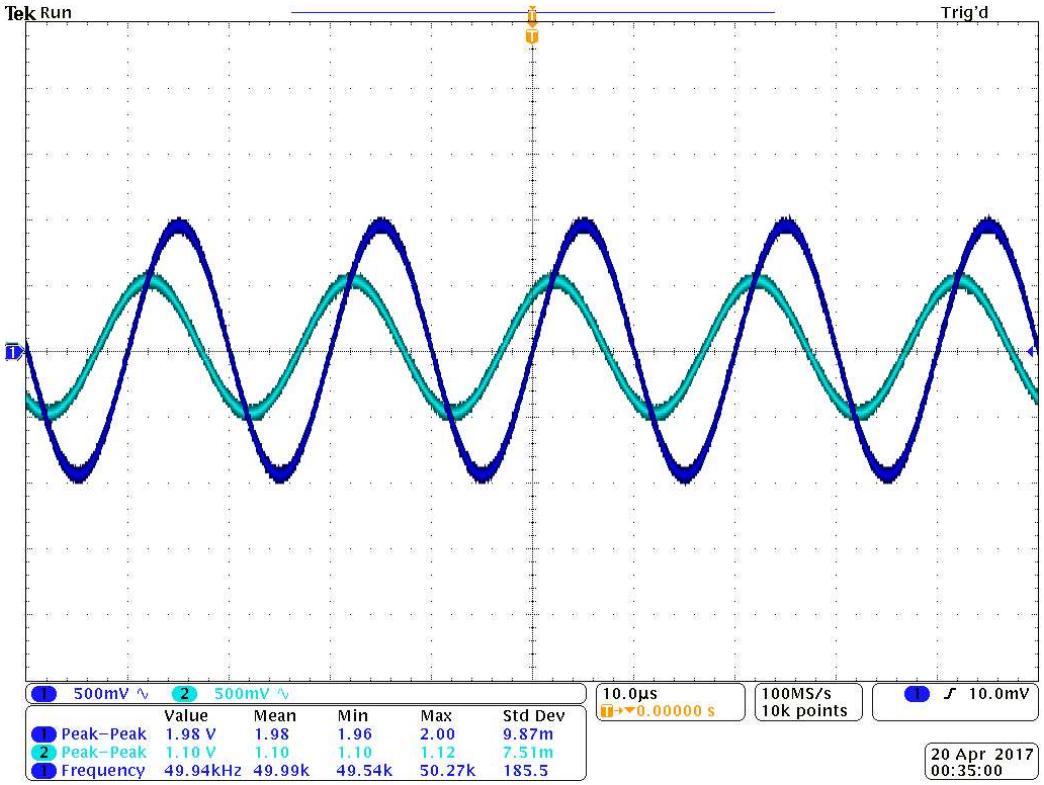


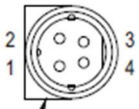
Figure 6.3.4: Prototype Filter Response at 50 kHz

6.4 Power Supply Testing

Shown below in figure 6.4.1 is the pin configuration for the central power supply unit. Measuring the voltage over pins 1/4 and 2/3 should result in a measurement of 20V. There should be no more than 180 mV peak-to-peak ripple voltage. This power supply is also equipped with a LED indicator which illuminates when the power supply is active.

Standard plug (all models) :

power DIN 4 pin with lock type, KYCON KPPX-4P equivalent

R7B		
	PIN NO.	OUTPUT
	1,4	+V
	2,3	-V

AC FG

-V connected to AC FG

Figure 6.4.1: Central Power Supply Pin Configuration

Each linear regulator can be easily tested in respect to its output voltage, dropout voltage, and line regulation to ensure it is properly working. The output voltage should be tested with an input voltage of 20 V and by measuring the output voltage. Dropout voltage can be tested by monitoring the output voltage while the input voltage is lowered slowly from 20 V to where the linear regulator starts to drop outside its intended voltage range. The input voltage where the drop starts is the dropout voltage. Lastly, the line regulation can be measured by taking the difference in output voltages for two different input voltages that are above the dropout voltage. Shown below in tables 6.5.1 and 6.5.2 are the testing parameters for each regulator and the tested results done for our breadboard prototype with no load.

Linear Regulator	Min-Max Regulated Output Voltage (V)	Maximum Dropout Voltage (V)	Maximum Line Regulation (mV)
LT1129IT-3.3	3.25-3.4	0.70	30
LD1085V50	4.9-5.1	1.5	10
L7815AB	14.4-15.6	2	150

Table 6.4.1: Linear Regulator Specifications

Linear Regulator	Regulated Output Voltage (V)	Dropout Voltage (V)	Line Regulation (mV)
LT1129IT-3.3	3.3013	0.128	1.22
LD1085V50	4.99153	0.844	2.03
L7815AB	15.132	0.83	97

Table 6.4.2: Prototype Testing Results

6.5 Software Design Testing

6.5.1 Anemometer Data from ADC

To test the anemometer data and the communication through the ADC, we will begin by putting the anemometer outside and then we will verify that we are able to receive values through the ADC to the Raspberry Pi and to verify that the value changes in real time according to the current conditions. Next, we will verify accuracy by comparing the current values from the anemometer to those collected by a separate instrument. After we have verified that the Raspberry Pi can receive values from the anemometer and that they are

accurate, we will verify how the anemometer behaves within the weather polling function and that the calculations done for wind speed and direction are accurate. To set up this part of the test, we will leave the anemometer outside and collect values for a particular time period. From those values, we will calculate by hand what the average speed and wind direction should be and compare it to the result from the corresponding logic in the weather polling function. Finally, once we have verified the Raspberry Pi can communicate with the anemometer, the readings are accurate, and the calculations are accurate, we will verify that those values are being stored correctly into the current weather object through print statements and the debug functionality of our IDE.

6.5.2 Temperature, Humidity, and Pressure Data

To test the communication between the MS860702BA01 temperature, humidity, and pressure sensor and the Raspberry Pi over the I2C bus, we will first individually poll the sensor for each weather reading, and compare it to the values reported by a separate weather reporting instrument to verify accuracy. After we have verified communication between the sensor and the Raspberry Pi at a foundational level, we will test communication through the weather polling function. We will do accomplish this by verifying that the values are only read at the specific intervals set by the loops and that the data is successfully stored in the current weather object. To do this, we will utilize print statements and the debugging functions of our IDE so that we can verify when the values are collected for each condition, how the value compares to current conditions reported by another instrument, and that the values are stored correctly and accessible. From here we will move onto testing how the weather is reported back to the pilot.

6.5.3 Weather Reporting

To test the weather reporting system, we will begin by testing the audio synthesis from the weather data to make sure the audio is not choppy and the correct values are being reported. After the audio has met our standards for quality and is consistently reporting correctly the values the function has been given, then we will move on to test the inputs to the function by making sure that any values that are received from the weather polling function are accurately reported. After this has been verified, we will move on to test the transmission.

We will begin testing the transmission by making sure the synthesized audio file can be played back to the radio. After we are sure that the function can communicate with the radio, we will verify how the system responds when the line is busy. We will make sure the system detects traffic on the line and waits to transmit until after there are no carrier signals detected.

6.5.4 ADS1015 ADC Channel

To test the ADS1015's conversion of Analog signal to digital signal, after soldering the header pins to the breadboard, we used the chips I2C protocols for transmitting the analog readings. Fortunately, Adafruit Industries provides great documentations and excellent open source python libraries. The functions of these libraries allowed us to read values from the ADS1015 using the I2C bus. Before starting, we must connect the Raspberry Pi with the ADC converter correctly. The table below shows how this connection is done.

ADS1015	**Connection Test**
VDD	3.3V (Pin1 of Pi)
GND	GND (Any Ground Pin of Pi)
SCL	SCL (Pin 5 of Pi)
SDA	SDA (Pin 3 of Pi)
Channel A0	To Middle Pin of Variable Resistor

Table 6.5.4.1 Wiring Test for Raspberry Pi and ADS1015

We proceed by connecting the 3.3-volt pin of the Raspberry Pi (pin1) to the VDD pin of the analog-to-digital converter. Then we connect the rest, the ground pin to any ground pins of the PI and the SCL pin to pin 5 of the Pi. The SCL provides the clock for all the peripheral devices when using the SDA connection. The SDA pin is connected to the 3rd pin on the Pi. We then use a potentiometer which is essentially a variable resistor that is used to test the ADS1015's channel port. The middle pin of the variable resistor can then be connected to any channels (we chose channel A0). After wiring the Raspberry Pi correctly with the ADS1015 connected to a breadboard, we proceeded with the software connection to the I2C bus. Before using the I2C bus it must be enabled on the Raspberry Pi after which, a couple libraries need to be installed as documented in the Adafruit's website for the ADS1015. After which we are can turn the dial on the potentiometer which changes the voltage coming into channel 0 of the ADS1015. The calculation and hard parts are all done by the ADS1015 libraries making it and easy to receive and manipulate the signals obtained by the analog sensors. An actual image of the wiring between the Raspberry Pi and ADS1015 is provided in figure 6.5.4.2 below.

7. Management

7.1 Task List

TASK	Estimated Completion Date	Person Responsible	Person Backup	Completed
Documents				
Task List Initial Draft	2/17/2017	Michael	Joshua	**Yes**
Hardware Block Diagram	2/17/2017	Joshua	Michael	**Yes**
Software Block Diagram	2/17/2017	Vanessa	Gilbert	**Yes**
Divide 60p Senior Design 1 Document Assignments	3/12/2017	Michael	Team	**Yes**
60p Senior Design 1 Document	3/30/2017	Michael	Team	**Yes**
Bill of Parts	7/10/2017	Michael	Team	**Yes**
100p Senior Design 1 Document	4/14/2017	Michael	Team	**Yes**
Final 120p Senior Design 1 Document	4/27/2017	Michael	Team	**Yes**
Radio				
Research and Determine Radio for Purchase	2/16/2017	Joshua	Michael	**Yes**
Purchase Radio	2/16/2017	Joshua	Team	**Yes**
Study Radio Schematic for Tieoff Locations	2/21/2017	Joshua	Michael	**Yes**
Confirm Locations of Critical Features in Lab	2/24/2017	Joshua	Michael	**Yes**
Correlate AGC Voltage to 3dBm Increments	3/17/2017	Joshua	Michael	**Yes**
Determine Audio Tx Voltage Level Needed	3/17/2017	Joshua	Michael	**Yes**
Determine Ideal Squelch Setting & Permanently Set Potetiometer	3/17/2017	Joshua	Michael	**Yes**
Design Audio Rx Input Circuit for Appropriate Mic Biasing Level	3/17/2017	Michael	Joshua	**Yes**
Design PTT Circuit	3/17/2017	Michael	Joshua	**Yes**
Research and Determine Permanent Connection	3/30/2017	Michael	Joshua	**Yes**
Purchase Cable/External Connectors	4/18/2017	Michael	Joshua	**Yes**
Modify Radio Case and Attach Connector	4/30/2017	Michael	Joshua	**Yes**
Carrier Detect				
Research and Design Schematic Using Comparator w/ Squelch Voltage	3/24/2017	Joshua	Michael	**Yes**

Order Parts	4/14/2017	Joshua	Team	**Yes**
Breadboard and Confirm Operation	4/24/2017	Joshua	Michael	**Yes**
Weather Sensors				
Determine What Measurements Will Be Collected	2/21/2017	Joshua	Michael	**Yes**
Make Decision on Sensors for Purchase	2/28/2017	Michael	Joshua	**Yes**
Research and Decide on Interfacing for Sensors	3/2/2017	Michael	Joshua	**Yes**
Design Annemometor Wind Speed Interfacing Circuit	3/10/2017	Joshua	Michael	**Yes**
Design Annemometor Wind Direction Interfacing Circuit	3/10/2017	Joshua	Michael	**Yes**
Buy Components for Breadboarding the Interfacing Circuits	3/10/2017	Michael	Team	**Yes**
Correlate ADS1015 Sensor Data w/ Wind Speed + Direction	4/14/2017	Gilbert	Vanessa	**Yes**
Correlate MS8607 Sensor Data w/ Temp, Humidity, Pressure	4/14/2017	Vanessa	Gilbert	**Yes**
Decide on Enclosure & Connection for MS8607	3/31/2017	Vanessa	Michael	**Yes**
Final Confirmation of Correct Operation	4/18/2017	Vanessa	Michael	**Yes**
Power Supply				
Determine What Voltages are Needed	3/17/2017	Joshua	Gilbert	**Yes**
Research and Design Voltage Regulation from 13.8 V	3/24/2017	Joshua	Michael	**Yes**
Determine Interfacing for 13.8V Tieoff	3/30/2017	Joshua	Michael	**Yes**
Order Parts	4/14/2017	Michael	Team	**Yes**
Breadboard and Confirm Operation	4/24/2017	Joshua	Michael	**Yes**
Interface Board				
1st PCB Design	5/13/2017	Joshua	Michael	**Yes**
1st PCB Order	5/14/2017	Joshua	Team	**Yes**
1st PCB Test	5/29/2017	Michael	Joshua	**Yes**
2nd PCB Design	6/19/2017	Joshua	Michael	**Yes**
2nd PCB Order	6/21/2017	Joshua	Team	**Yes**
2nd PCB Test/Confirm Operation	7/5/2017	Michael	Joshua	**Yes**
Microcontroller				
Research and Decide on MCU	3/2/2017	Gilbert	Michael	**Yes**

Research and Decide on Appropriate ADC	3/2/2017	Gilbert	Michael	**Yes**
Order MCU	3/2/2017	Gilbert	Team	**Yes**
Order ADC	3/2/2017	Gilbert	Team	**Yes**
Software				
Decide on Operating System for MCU	3/10/2017	Gilbert	Vanessa	**Yes**
Draft Word Bank for AWOS Standard Reporting	3/17/2017	Joshua	Gilbert	**Yes**
Research and Decide on Voice Library	3/17/2017	Gilbert	Vanessa	**Yes**
Create Comprehensive Logic Diagram for Decision Making/Operation	3/17/2017	Gilbert	Vanessa	**Yes**
Create Python Library for I2C w/ ADS1015	3/31/2017	Gilbert	Vanessa	**Yes**
Create Python Library for I2C w/ MS8607	3/31/2017	Vanessa	Gilbert	**Yes**
Enable Raspberry Pi for CODEC Communication	3/31/2017	Michael	Vanessa	**Yes**
Write Function to Correlate AGC Voltage to Received Signal Power	4/14/2017	Gilbert	Vanessa	**Yes**
Write Program for Weather Measurement Logic	4/15/2017	Vanessa	Gilbert	**Yes**
Write Function for Audio Rx Recording	4/16/2017	Gilbert	Michael	**Yes**
Write Function for Audio Tx to CODEC (w/ PTT)	4/17/2017	Gilbert	Michael	**Yes**
Program Raspberry Pi for Main Logic Tree	4/30/2017	Vanessa	Gilbert	**Yes**
Design Website Interface w/ Remote Access	SD2 TBD	Vanessa	Gilbert	**Yes**
Optional Mounting Case				
Research and Decide Viability of 3D Printing				
Decide on Location for Printing				
Create Model 3D Design with known PCB Dimensions				
Slice 3D Drawing				
Print First Prototype				
Make Revisions				

7.2 Budget

Item	Design Quantity	Backup Quantity	Engineering Justification and Notes	Estimated Expense
Anemometer	1	0	Wind speed and direction sensor. Provided by Mr. Young.	$0.00
Barometer	1	0	Atmospheric pressure sensor	$10.00
Hygro Thermometer	1	0	Temperature and humidity sensor	$35.00
ADC	1	3	Receive analog data, convert analog to digital data, process digital data	$10.00
Raspberry Pi 3	1	1	Process pilot voice and commands, provide data to website	$60.00
Operational Amplifiers	3	6	Audio conditioning	$15.00
Comparator	1	3	Carrier detect	$5.00
Diodes	1	4	University lab kit.	$0.00
Transistors	10	10	Weather instrument signal processing and audio conditioning	$10.00
Linear Regulators	3	6	Convert 20 V power supply to lower voltage for different stages	$27.00
Other ICs	N/A	N/A	For possible future use system	$10.00
Ports/Headers	N/A	N/A	Supply correct and secure connections	$5.00
PCB + Labor	1	1	Fabricate PCB and install components	$120.00
General Passive Components	N/A	N/A	General resistors, inductors, capacitors for various parts of design.	$10.00
Power Supply	1	1	Provide power for aviation radio and system	$30.00
Aviation Radio	1	0	Used to transmit and receive signals to and from system to pilot	$55.00
Total	-	-	--	$402.00

Table 7.2: Budget Allocation

7.3 Milestones

Project Tasks	Design Milestone	Order Milestone	Test Milestone	Final Design Revision and Test Milestone
Weather Instruments	N/A	02/28/2017	03/07/2017	04/18/2017
Weather Instrument Analog System	03/15/2017	03/17/2017	03/27/2017	04/18/2017
Power Supply System	04/01/2017	4/03/2017	04/10/2017	04/18/2017
Audio System	03/15/2017	03/17/2017	3/27/2017	04/18/2017
µC/DSP/CPU	N/A	02/28/2017	03/07/2017	04/18/2017
Webpage		N/A		04/18/2017
Digital Weather Reporting	03/22/2017	N/A	3/30/2017	04/18/2017
Digital Communications Check	03/18/2017	N/A	3/30/2017	04/18/2017
1st Prototype	N/A	N/A	05/03/2017	05/06/2017
1st PCB	05/13/2017	05/14/2017	N/A	05/29/2017
2nd Prototype	06/05/2017	06/06/2017	N/A	06/13/2017
2nd PCB	06/19/2017	06/21/2017	N/A	07/05/2017
60 Page SD1 Design Draft (15 Pages/Person)	N/A	N/A	N/A	3/31/2017
100 Page SD1 Design Draft (25 Pages/Person)	N/A	N/A	N/A	4/14/2017
Final 120 Page SD1 Design Draft (30 Pages/Person)	N/A	N/A	N/A	4/27/2017

Table 7.3; Milestones with Deadlines

8. Appendix

8.1 Datasheets

8.1.1 Raspberry Pi

https://www.raspberrypi.org/documentation/hardware/computemodule/RPI-CM-DATASHEET-V1_0.pdf

8.1.2 AWOS

http://www.coastalenvironmental.com/aviation-weather-stations.html

8.1.3 UHF/VHF Range Calculations

http://arundale.com/docs/ais/AppNote_UHF_VHF_Calc.pdf

8.1.4 ADS101x-Q1

http://www.ti.com/lit/ds/symlink/ads1013-q1.pdf

8.1.5 Audio CODEC Proto

https://download.mikroe.com/documents/add-on-boards/other/audio-and-voice/audio-codec-proto/audio-codec-proto-manual-v100.pdf

8.1.6 WM8731 CODEC

http://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf

8.1.7 Low Drop Power Schottky Rectifier

http://www.st.com/content/ccc/resource/technical/document/datasheet/d8/3f/72/85/bc/90/4e/f7/CD00001626.pdf/files/CD00001626.pdf/jcr:content/translations/en.CD00001626.pdf

8.1.8 TVS Diode Arrays

http://m.littelfuse.com/~/media/electronics/datasheets/tvs_diode_arrays/littelfuse_tvs_diode_array_sp4020_datasheet.pdf.pdf

8.1.9 Low Noise Op Amp

http://www.ti.com/lit/ds/symlink/sa5534a.pdf

8.1.10 Micropower Low Dropout Regulator

http://cds.linear.com/docs/en/datasheet/112935ff.pdf

8.1.11 Positive Voltage Regulator

http://www.st.com/content/ccc/resource/technical/document/datasheet/41/4f/b3/b0/12/d4/47/88/CD00000444.pdf/files/CD00000444.pdf/jcr:content/translations/en.CD00000444.pdf

8.1.12 120W AC-DC Adaptor

http://www.alliedelec.com/m/d/c6fd3490220d0ac225701a0cd2276943.pdf

8.1.13 Step Down Voltage Regulator

http://www.ti.com/lit/ds/symlink/lm2676.pdf

8.1.14 PD-40S

http://www.cui.com/product/resource/pd-40s.pdf

8.1.15 IC-A2 Maintenance Manual

http://www.repeater-builder.com/icom/pdfs/ic-a2-maint-man.pdf

8.1.16 IC-A2 Owner's Manual

http://radiopics.com/1.%20Manuals/Icom/Icom-Air/Operation%20Manuals/Icom_IC-A2%20(Owner%27s%20Manual).pdf

8.1.17 Anemometer

http://www.davisnet.com/product_documents/weather/spec_sheets/7911_SS.pdf

8.1.18 MS8607-02BA01

http://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data+Sheet%7FMS8607-02BA01%7FB%7Fpdf%7FEnglish%7FENG_DS_MS8607-02BA01_B.pdf%7FCAT-BLPS0018

8.2 Software

8.2.1 Raspberry Pi/ADC

https://learn.adafruit.com/raspberry-pi-analog-to-digital-converters/ads1015-slash-ads1115

8.2.2 SMBus

https://pypi.python.org/pypi/smbus2/0.1.2

8.2.3 Raspberry Pi/I2C

http://www.raspberry-projects.com/pi/programming-in-python/i2c-programming-in-python/using-the-i2c-interface-2

8.2.4 PicoPi

https://github.com/DougGore/picopi

8.2.5 Python Style Guide

https://www.python.org/dev/peps/pep-0008/