System Specification

Team Brew Science

By: Reagan Stovall, Andrew Klonitsko, Andrew Gates

Version: 2.0

Revision History Block

Version 0.1: Changes made include,

- Initialization and formatting of all portions of the document

Version 1.0: Changes made include,

- Each topic and subsequent topics of the System Test Plan has been filled out
    - 1. Purpose
    - 2. Scope
    - 3. System Overview
    - 4. System Architecture
    - 5. Human Interface Design
    - 6. Testing
    - 7. Timeline
    - 8. References

Version 2.0: Changes made include,

- Modified sections to new information where applicable.
- Added new diagram for schematic of Raspberry Pi Shield PCB.
- Added new diagram for sensor configurations.
- Added in datasheets for currents parts in use in References.

Version 3.0: Changes made include,

- Modified sections to finalized information where applicable.
- Updated diagrams and datasheets for final setup.

1. Purpose

1.1. Introduction

This section is designed to explain what the purpose of this document is. It will detail what the purpose of a specification document is, what it describes, and who the intended audience is.

1.2. What is the point of this specification? What does it describe? Who is the intended audience

The purpose of this specification is to give the reader an overview of everything that is involved in our project. It describes in depth all about the scope of our project, an overview of the system, the system architecture used, how it interfaces with the user, and how it was tested. The intended audience is anyone who desires to use our project, or wants to learn more about what was involved in creating this project.

2. Scope

2.1. Introduction

This section is designed to explain the overall system. It will describe what this system is, what it is designed to do, how it interacts with the user/environment, and what the benefits of this device are. It will also explain what this overall specification document will cover.

2.2. Identify product to be described. What does this specification cover?

The product that will be described is a device that can be used as an automation station. This device allows for measuring and controlling of various devices. This specification covers everything involved in our system from the hardware components, to the software interface, to the database portion, as well as the interaction between these systems. It also covers testing and possible issues that may arise when in use.

2.3. Explain what the overall system does and how it interacts with users or environment.

Our device is designed to allow a wide range of users to measure and control various sensors, motors, actuators, and etc, through the use of a simple and easy to follow interface. The user is not required to know much about programming in order to use this device, the device is made to be as user friendly as possible to allow for even the average person with little technological experience to use this device. The user can create, save, and run various sensor and control configurations using a broad range of sensors that are applicable with our device.

2.3.1. Describe the benefits of the system.

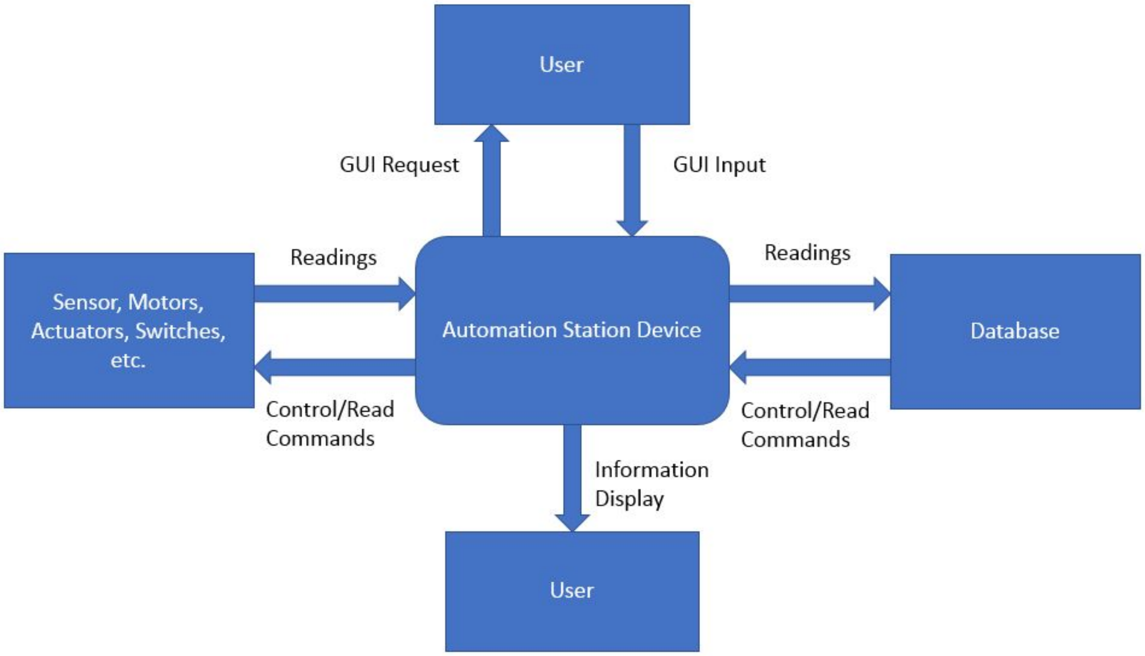
The benefits of this system are to allow users to monitor and control various systems such as home breweries, home wineries, aquaponic systems, and etc. The benefit of our device is that the user isn't required to have any technology or coding experience. Any user can get a generic sensor and connect it to our device and start reading this sensor in minutes. From there the user can also connect motors, actuators, and switches to be activated when sensors reach a certain level. This allows the user to control anything in their life that they desire with ease.

3. System Overview

3.1. Introduction

The purpose of this section is to explain the system in a high level view. It will provide an overview for the system, purpose, context and background for the system as well.

3.2. Introduce the system context.



3.3. Provide context and background for the project/system.

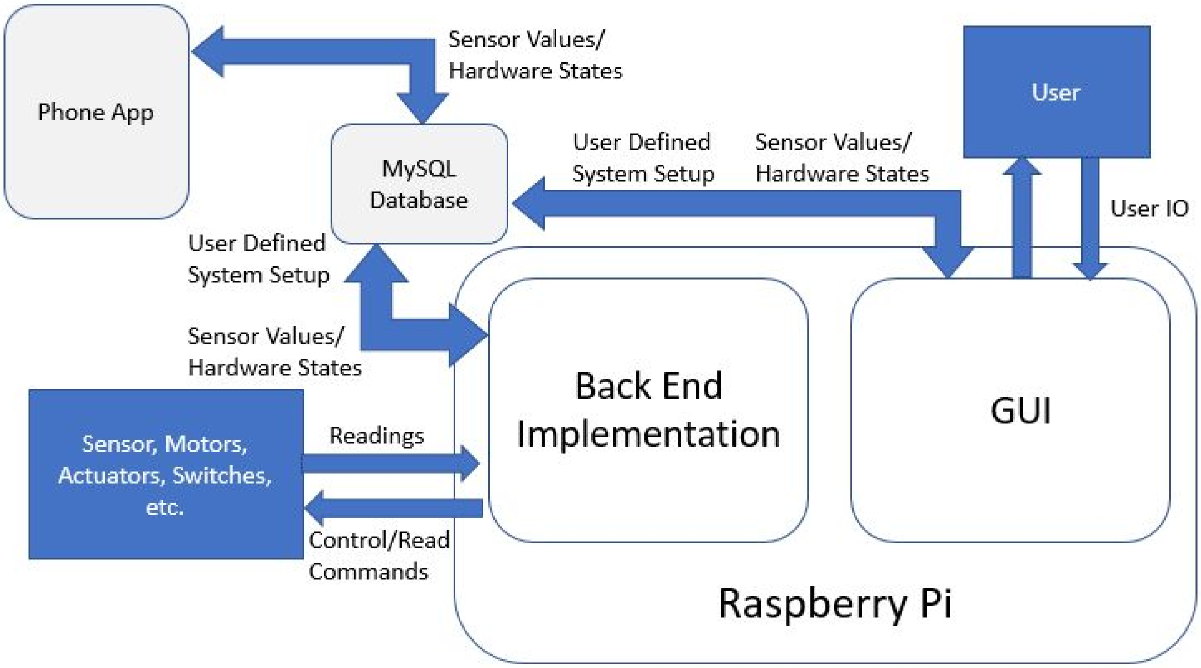
When meeting with customers involved in breweries, we realized that a lot of them were using little to no automation and control in their brewing process. This was due to most of them being very knowledgeable brewers, but having little knowledge about how to properly control their systems. There are many automation systems that can be implemented breweries, wineries, and etc, but all of these require some depth of knowledge about technology, automation, electrical/computer engineering, and coding. Our device is designed to address all of those issues and help users who want to automate or monitor aspects of their life no matter what their skill level is.

4. System Architecture

4.1. Introduction

This section outlines everything involved in the architecture side of our project. From the system diagram, to the components, to the interactions themselves, this section will outline everything that is used to create our project and how everything intertwines.

4.2. Include a diagram that shows at a high level the entire system



4.3. If there are major components that require additional detail, break those out and sequence them in relation to the overall architecture diagram.

Raspberry Pi: The Raspberry Pi is a credit-card-sized computer that plugs into your TV and a keyboard. It is a capable little computer which can be used in electronics projects, and for many of the things that your desktop PC does, like spreadsheets, word processing, browsing the internet, and playing games. It also plays high-definition video. We want to see it being used by adults and children all over the world to learn programming and digital making. Retrieved from https://www.raspberrypi.org/help/faqs/

Multiplexers: These components allow for multiple sensors to connect to a chip, and then based on the signal that you give the chip, the desired sensor signal will come through the output. We are using 16-to-1, 8-to-1 and 2-to-1 multiplexers in our design. This will allow for the selection of which sensor to read, which voltage to use, which resistor to use, and so on.

4.3.1. Describe the architecture as shown in your architecture diagrams.

The architecture in our architecture diagram outlines the main connections between every aspect of our system. There are really three main components. One of such being the interface between the user and the GUI. Another being the connection between what settings the user produces through the GUI to the sensors, motors, actuators, switches, and etc. The last being the connection between these components to the database to be viewed. All of these connections go both ways to send and receive data from the user, the components, and the database.

4.4. Describe alternative designs that you considered and describe the benefits or drawbacks of each

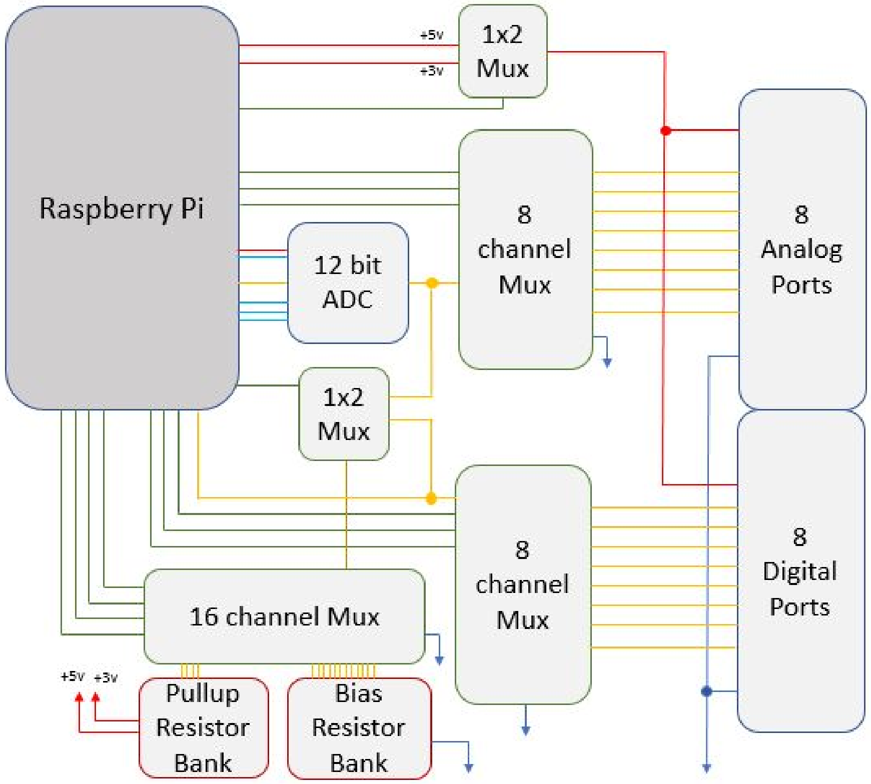
This was the main design that we considered from the start, our overall system has not changed much but some of the individual components have.

- We experimented with using various transistors at first instead of multiplexers. We ran into issues with transistors though which arose from using multiple different transistor types in the same section in order to properly control the direction of the circuit.
- We originally were planning on using one 16-to-1 multiplexer to allow for 8 digital and 8 analog sensors to connect to it. We ran into issues though with deciding how to control the output of this multiplexer depending on what type of sensor and how many pins it had. We decided to switch to 2 different 8-to-1 multiplexers and make one devoted to analog and one devoted to digital.
- We considered a transistor array initially to control the bias resistors. This had drawbacks because we were only able to interface a couple resistors but not control properly whether they were pull-up with 3.3 volts, pull-up with 5 volts, or pull-down. We decided to switch this to a 16-to-1 multiplexer to allow for 8 different pull-down bias resistors, 4 different pull-up bias resistors with 3.3 volts, and 4 different pull-up bias resistors for 5 volts.
- For the GUI package, we researched other packages for GUI creation. We found that some were either too high level for what we were trying to do, or too low level. Tkinter was the perfect balance of usability and customizability for what we were trying to achieve.

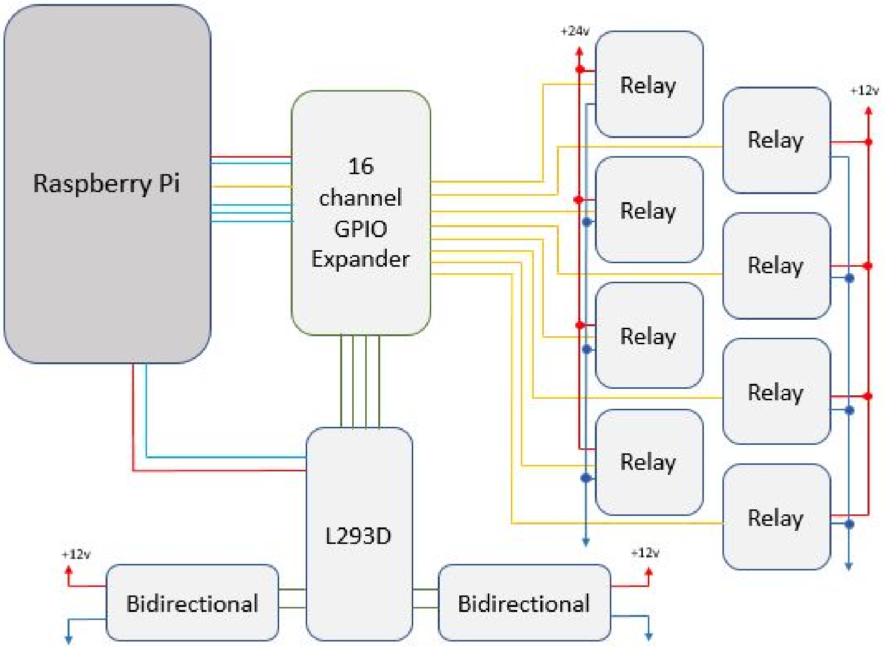
4.5. Describe the rationale for why you decided on the described design.

The main aspects that we were trying to reduce in creating our design was GPIO pins, space, and power consumption, while trying to maintain flexibility and usability.. This design, by using multiplexers in conjunction with the ADC chip, and various resistors, allowed us to have many different combinations of resistor types that can be used, while minimizing GPIO pins, space, and power consumption. On the GUI side we used tkinter which is a robust package that allows for the creation of very user friendly GUI systems, which is one of our main goals.

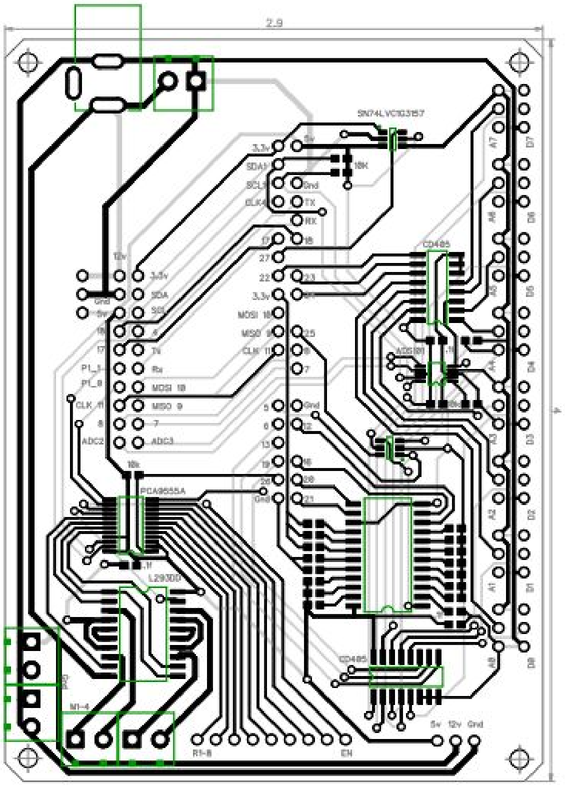
4.6. Use one complete section to describe the hardware design you used for the system. Include block diagrams for IP blocks, wiring diagrams, or circuit layouts developed. Remember that this section should have sufficient detail to reproduce your system.



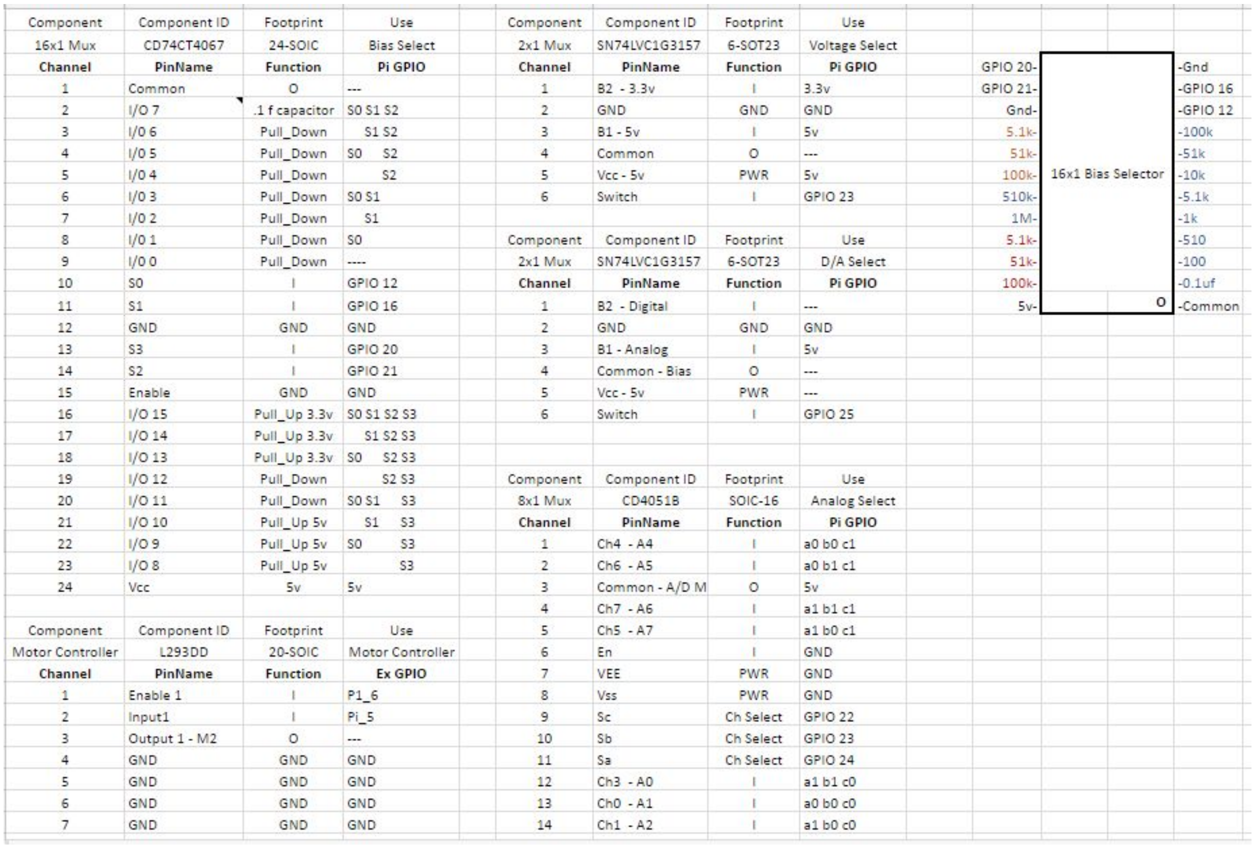
Sensor Schematic V2.0



Hardware Schematic V1.1



Raspberry Pi Shield PCB



Component	Component ID	Footprint	Use
16x1 Mux	CD74CT4067	24-SOIC	Bias Select
Channel	**PinName**	**Function**	**Pi GPIO**
1	Common	O	---
2	I/O 7	.1 f capacitor	S0 S1 S2
3	I/0 6	Pull_Down	S1 S2
4	I/0 5	Pull_Down	S0 S2
5	I/0 4	Pull_Down	S2
6	I/0 3	Pull_Down	S0 S1
7	I/0 2	Pull_Down	S1
8	I/0 1	Pull_Down	S0
9	I/0 0	Pull_Down	----
10	S0	I	GPIO 12
11	S1	I	GPIO 16
12	GND	GND	GND
13	S3	I	GPIO 20
14	S2	I	GPIO 21
15	Enable	GND	GND
16	I/O 15	Pull_Up 3.3v	S0 S1 S2 S3
17	I/O 14	Pull_Up 3.3v	S1 S2 S3
18	I/O 13	Pull_Up 3.3v	S0 S2 S3
19	I/O 12	Pull_Down	S2 S3
20	I/O 11	Pull_Down	S0 S1 S3
21	I/O 10	Pull_Up 5v	S1 S3
22	I/O 9	Pull_Up 5v	S0 S3
23	I/O 8	Pull_Up 5v	S3
24	Vcc	5v	5v

Component	Component ID	Footprint	Use
Motor Controller	L293DD	20-SOIC	Motor Controller
Channel	**PinName**	**Function**	**Ex GPIO**
1	Enable 1	I	P1_6
2	Input1	I	Pi_5
3	Output 1 - M2	O	---
4	GND	GND	GND
5	GND	GND	GND
6	GND	GND	GND
7	GND	GND	GND

Component	Component ID	Footprint	Use
2x1 Mux	SN74LVC1G3157	6-SOT23	Voltage Select
Channel	**PinName**	**Function**	**Pi GPIO**
1	B2 - 3.3v	I	3.3v
2	GND	GND	GND
3	B1 - 5v	I	5v
4	Common	O	---
5	Vcc - 5v	PWR	5v
6	Switch	I	GPIO 23

Component	Component ID	Footprint	Use
2x1 Mux	SN74LVC1G3157	6-SOT23	D/A Select
Channel	**PinName**	**Function**	**Pi GPIO**
1	B2 - Digital	I	---
2	GND	GND	GND
3	B1 - Analog	I	5v
4	Common - Bias	O	---
5	Vcc - 5v	PWR	---
6	Switch	I	GPIO 25

Component	Component ID	Footprint	Use
8x1 Mux	CD4051B	SOIC-16	Analog Select
Channel	**PinName**	**Function**	**Pi GPIO**
1	Ch4 - A4	I	a0 b0 c1
2	Ch6 - A5	I	a0 b1 c1
3	Common - A/D M	O	5v
4	Ch7 - A6	I	a1 b1 c1
5	Ch5 - A7	I	a1 b0 c1
6	En	I	GND
7	VEE	PWR	GND
8	Vss	PWR	GND
9	Sc	Ch Select	GPIO 22
10	Sb	Ch Select	GPIO 23
11	Sa	Ch Select	GPIO 24
12	Ch3 - A0	I	a1 b1 c0
13	Ch0 - A1	I	a0 b0 c0
14	Ch1 - A2	I	a1 b0 c0

	16x1 Bias Selector	
GPIO 20-		-Gnd
GPIO 21-		-GPIO 16
Gnd-		-GPIO 12
5.1k-		-100k
51k-		-51k
100k-		-10k
510k-		-5.1k
1M-		-1k
5.1k-		-510
51k-		-100
100k-		-0.1uf
5v-	O	-Common

Sensor Configurations

As shown in our diagram our system involves various multiplexers, resistors, and paths that can be used to control the flow of current for the desired sensor setup, and everything is attached to the Raspberry Pi.

- For the 2 different voltage levels that can be selected there is a 2-to-1 multiplexer to choose whatever the desired voltage is.
- For the 2 different types of sensors (analog or digital) there are 8 pins for each for a total of 16 different sensors.
- There is a 16-to-1 multiplexer to choose from various different resistors setups, 8 being pull-down with different resistors, 4 being pull-up with different resistors and 3.3 volts, and 4 being pull-up with different resistors and 5 volts
- The 8 analog pins go straight through the ADC chip for 3 pins, or to the 16-to-1 multiplexer for 2 pins.
- The 8 digital pins go straight to the 16-to-1 multiplexer which then goes to any of the 16 different resistor combinations.
- Finally there are pinouts to allow for various other type of sensors that require I2C to be used.

4.6.1. Use one complete section to describe the software design you used in your system. Remember that this section should have sufficient detail to reproduce your system.

Software was divided into 4 main parts for this project.

1. The GUI: This was designed for the purpose of retrieving the necessary data from the user to implement a control system, then display the data from sensors and hardware in a realtime environment. The data required from the users is as follows:
    a. Analog Sensors:
        i. Port
        ii. Type
        iii. Name
        iv. Number of Pins
        v. Bias Resistor needed depending on the Number of Pins
        vi. The voltage supplied to the sensor, either 3.3v or 5v
        vii. Sensor Name
        viii. Calibrate
    b. Digital Sensors:
        i. Port
        ii. Type
        iii. Name
        iv. The voltage supplied to the sensor, either 3.3v or 5v
        v. Sensor Name
        vi. Calibrate
    c. Motors:
        i. Port
        ii. Motor Name
        iii. Speed
    d. Relay:
        i. Port
        ii. Relay Name
    e. I2C:
        i. Sensor Type
        ii. Sensor Name
        iii. Calibrate
    f. Timing Choice:
        i. Continuous

        ii. Periodic
        iii. One-Shot
    g. Hardware/software interactions as a function [Sensor, Value, Hardware, Action, Start, Duration, End]
2. The BackEnd: This is the program that reads the data collected by the GUI and implements the functions given. Each function is read in during the time frame it is active and the program responds accordingly. Along with handling all of the various aspects of the GPIO management, it will also report the values of the sensors and states of the hardware to a database every minute for long distance monitoring.
3. The Database: This is a background system run on MySQL and hosted on a UTW server. It's implemented to ease data management and lock issues between the various processes.
4. Phone App: At this time, the only application of the Phone app is monitoring the sensors and data. Multiple sensors and hardware options will be available for checkbox selection, at which point the a screen displaying a graph for each will appear. At a later date, a full app that can set up and implement the BackEnd will be available for the phone so that the raspberry pi can operate headless, reducing the overall cost by roughly 50%. **assumes user has access to smart phone*

4.6.2. Describe how your system components interact.

Initially our software GUI will allow the user to create sensor configuration settings. This will include the name of the sensor, the type of sensor, the type of voltage, the number of pins the sensor needs, and the type of resistor if required. Once the user makes all of these settings, finalizes them, and runs the program, they will be sent to another program to run the system. This program will interface the settings that are sent over through multiple arrays, and based on the desired settings for each sensor, it will set up that desired path in our circuit. Once the desired path has been set up, and if the timing conditions are satisfied, the sensor will be read, the output will be saved to a textfile, which will then be uploaded to our database. Once it is on the database a user will be able to view these outputs from their phone.

4.6.3. Describe protocols you developed to enable interaction

For the GUI we used the tkinter libraries to create the interface the user will see to create, save, and run their configurations. For the program that will run the sensors we used RPi to interface the various GPIO sensors to be controlled.

4.8. Describe key algorithms used within your design and how they work. Describe why you chose these algorithms over alternatives.

No algorithms are used yet. Nothing at this point can be optimized or needs to be optimized with more efficient algorithms yet.

4.9. Describe dependencies your project relies on.

Hardware: For the hardware portion the main dependency is with the multiplexers. These are the core of our project and without them we would have to use 35+ transistors to control every signal that is going through our circuit. The multiplexers allow us to reduce GPIO pin usage, reduce power consumption, and reduce space.

GUI: The dependencies of the GUI are user capability. The GUI and project in whole is based on the premise that the user does not have coding or circuit design experience. This requires the GUI to allow a user to build a complex, multi functional control system with common language and simple button features.

Database/Phone: The dependencies of the phone app is having a database that is reliable and access to the internet; these are important because all the data that will be display to the user are going to be in real time. Another dependency is a simple user interface that the users will understand what they are looking at; without this, the app would be useless.

5. Human Interface Design

5.1. Introduction

This section is devoted to how the user will view and control our system. It involves use cases and how the user should expect to use this system.

5.2. Describe how the system works from the user's perspective.

From a user's perspective, the user will turn our system on which will bring them to our GUI software. Once they are here it will give them options for settings, or to create a new setup or resume an old setup. When creating a new setup the user will very easily be able to add sensors, define what voltage the sensor requires, what type of sensor it is, the number of pins it requires, and what resistor it requires if necessary. The user will be able to load up to 16 different sensor configurations with ease, and connect those sensors to various motors, actuators and switches with control conditions. The user will also have access to modify the timing of readings for each sensor with a simple menu. Once the user is done all they have to do is select run, which will create their desired sensor setup and start reading the sensor data, allowing them to view it from the GUI or from their phone.

5.3. Articulate the use cases and how the users should expect to use the system.

The main use case is -
User -> Enter info for sensor data into GUI -> Enter info for what the sensors will control -> Conform settings -> Run setup -> View sensor outputs.

The user should expect to use the system to set up their sensors, actuators, switches and motors through the use of the attached GUI. The user should expect to be able to create a new system configuration with ease and be able to edit all aspects of each sensor, and what said sensor will control, if anything. The user should then expect to be able to use these configurations to start a new process which they will then be able to view the outputs from the various sensor outputs.

6. Testing

6.1. Introduction

Testing is important to ensure that every single situation where an error could occur has been accounted for, and fixed. It shows that this item is, to the best of our knowledge, failsafe. Testing also allows for the user to see where issues may arise, so that they can compensate for such issues is any occur.

6.2. Describe how you tested the system.

Hardware: For the hardware portion of our project, we tested every single component individually first. We ensured that each component would work on it's own before adding in the next portion. We tested each component with various setups such as 3.3 V or 5 V, analog or digital, 2 pin or 3 pin, and with various bias resistors for pull-up and pull-down.

GUI: The GUI is somewhat standard in that it is a single container application with pages built as classes to be called when needed. Each page is built off a single template and filled with both common and unique features. For each new item created, a sample program was created to test and improve the function before it was then added to the Main GUI. The Goal is to be able to load predefined template programs and run them with little setup. We will be able to use this feature to test corner cases in the future.

Database/Phone: The database will be tested by uploading data from one computer and network to the database and download data from another computer and network. If the following works, the database will be consider to be functional and working properly. The phone app will be tested as each activity is built. Testing will consist of performing interrupts during the running of the program as well making sure all of the buttons perform the right task given. Testing of bad data will also happen to see how the software handles invalid data.

6.3. Describe the procedures necessary to test at relevant stages of development. For example, did you test individual hardware components before assembly? How? Did you unit test your software? How?

Hardware: We first made sure that we could select between 3.3 V and 5 V. Once we were able to do that we tested reading various sensors with changing voltages. Once we were able to do that we interfaced the sensor into a multiplexer, and so on and so forth. We built our circuit step by step, and never added new components until we thoroughly tested the current circuit with all different configurations.

GUI: We modularized each new widget, testing implementations and functions in stand alone environment, making sure that each part worked individually before adding to the whole. For data testing, we are currently writing to a txt file that will be read by the “Run Program” since no read/write collisions can take place at this time, no testing was done for this. When the database and on the fly changes are implemented, loaded templates will be used to test the data management system.

Database/Phone: We first test the database by creating an array with random data and serialize it and upload to the database. For the app we first test what happened when the program got interrupted. The second test is what happens when the program shuts down unexpectedly. The final test is how the user interface responds. We use adb to do most of our testing as well as using an android phone.

6.4. Describe test cases you used that are required to ensure full functionality.

The main test case that we used to ensure full functionality is a fully loaded system with every sensor, motor and actuator port full. This is the situation where the system will be at max stress. If it can work under this condition then it will work under any other situation so long as the user follows the same constraints for the type of sensors and motors.

The other test case that we used to ensure full functionality is testing the system with compatible and incompatible sensors and motors. As expected, the user should only use compatible sensors and motors since the incompatible ones will either result in false data, or no data.

6.5. Describe conditions for expected failure. Did you test those and how?

Hardware:

- One main issue is if the components overheat and burn out. This could be due to voltage levels switching too quickly, or signals being sent too quickly to the multiplexers. We tested doing so with no time delays and there was not a noticeable increase in heat, or decrease in performance by any of the components. We decided to introduce small delays between changing voltages and reading sensors to account for this issue if it happened to arise though.
- Another main issue is if components came detached from the display screen, the Raspberry Pi, or the PCB. To account for this expected failure, we enclosed our entire product in a case to protect all of the components and boards. We also soldered our PCB directly to the Raspberry Pi through some pin extenders to allow for maximum durability.
- Lastly another condition for expected failure would be if the user tries to attach a sensor that is not compatible, or perhaps broken. We tested these situations by using broken sensors and ones that we knew would not work with our setup. Sometimes we were able to detect this by the output values (all 0's, 1's, or same values). But sometimes we were unable to do so and the sensor would fluctuate just enough for us to be unable to detect this. If we are able to inform the user of a sensor that is not working we will do so, but failure still might slip through due to the consistency of sensors that are broken or non-compatible. If this is the case the user will have to check to make sure that the readings are all reasonable and expected values before continuing with said sensor.

GUI:

- Failure from the GUI will come from user experience. The functionality of the device is still somewhat limited despite all efforts to make global, but some of those limitations come from the GUI itself. If a user wants to make the device more complicated, they will eventually find the wall of what is possible. From the software side, the goal is for every sensor port, and every hardware port to be able to be utilized at the same time. A critical failure would be if that was not possible, but as we are still in the development stage, this is very difficult to test.
- If this critical function is met, the next test would be the user's experience with all of these functions at the same time. The "Display Page" will show the running applications in a graph with up to 3 chosen sensors and hardware components displayed at any given time. If this cannot display the readings of multiple sensors and hardware states continuously, then this too would be considered a failure.

Database/Phone:

- Failure from the database will come from if connection is lost during the transfer of data to the server. This will be tested by trying to send data to the server while not connected to the database. The database can also fail if we write from two different devices at the same time; this will be tested by having the raspberry pi and the app connect to the server at the same time trying to send data.
- Failure from the phone will come if another program interrupts our current program; this will be tested by having the app run and having someone else call the phone to cause the interrupt.The phone app can also fail if it can't retrieve data from our database; this will be tested by having the phone off line and trying to run the app.

7. Timeline

7.1. Introduction

This section is meant to be an overview of the timeline of development in order to ensure proper sequencing. A timeline is important so that small steps of progress can be seen and understood. The timeline shows exactly how we went from nothing to our end result.

7.2. Hardware:

- Read and deciphered the sensor settings arrays to be delivered from the GUI so that the proper circuit paths can be selected for various sensors through a Raspberry Pi.
- Designed circuit schematic to allow for various circuit selections based on the type of sensor (digital or analog), the type of voltage (3.3 or 5), the type of pin setup (3 or 3), and the type of bias resistor if required (10, 100, 1k, 10k, 100k ohms).
- Created control of input voltage (either 3.3 V or 5 V) through the use of transistors and a 2-to-1 multiplexers.
- Interfaced an 8-to-1 multiplexer to allow for 8 different analog sensor selections based on the input signal.
- Connected the output of the 8-to-1 analog multiplexer to an ADC converter, which will then be read by the Raspberry Pi. This will only work for 3 pin analog sensors.
- Interfaced an 8-to-1 multiplexer to allow for 8 different digital sensor selections based on the input signal. This will only work for 3 pin digital sensors.
- Interfaced a 16-to-1 multiplexer to allow for various bias resistor setups such as pull down, pull up 3.3 V and pull up 5 V
- After testing was complete, created a PCB to allow for finalization of our circuit schematic.

7.3. GUI:

- First we created a list of everything the GUI program had to display and return, and then we created "Pages" for each action.
    - Welcome Page: A simple opening screen that transitions to the application
    - Home Page: For deciding what the user wants to do app wise.
    - Sensor Setup Page: Pick sensor ports, types, bias resistors
    - Hardware Setup Page:Pick hardware components, and their associated ports
    - Timing Choice: Choose between Continuous, Periodic, OneShot cycles.
    - Timing: Retrieve dependencies for each hardware component.
    - Run: A simple page to initiate the background program
    - Display Page: A real time review page that displays a choice of multiple sensors, hardware states, and timeline all on a continuously updating graph.
- Next we implemented each page, individually, learning to use the software libraries.
- Data management was decided for simplicity initially using a txt file.

7.4. Database/Phone:

- First we created a database using the server provided by uwt and created a connection between the android app and the database.
- We created the database table and the php code to send the database an array with the data that we want to pass between devices
- Next we built the gui interface for the phone app, the following are the activity created:
  - Splash Screen: A simple screen that's up for a seconds with our team name
  - Home Page: For deciding what the user wants to do app wise
  - Live Graph: Display graph for the sensor that is chosen.
  - Status: Pick hardware components, and their associated ports
  - Stop: A simple activity to stop the program
- Next we implemented each activity, individually, learning to use the software libraries.
- Finally we will test the app and database trying all possible ways of breaking it.

8. References

https://www.raspberrypi.org/help/faqs/
12 Bit ADC - http://www.ti.com/lit/ds/symlink/ads1013.pdf
16x1 MUX - http://www.ti.com/lit/ds/symlink/cd74hc4067.pdf
8x1 MUX- http://www.ti.com/lit/ds/symlink/cd4051b-mil.pdf
2x1 MUX - http://www.ti.com/lit/ds/symlink/sn74lvc1g3157.pdf
12V to 5V - http://power.murata.com/data/power/oki-78sr.pdf
Motor Controller - http://www.ti.com/lit/ds/symlink/l293.pdf
GPIO Expander - http://www.nxp.com/documents/data_sheet/PCA9555A.pdf