eHVAC: Wireless Modular Multi-Zone HVAC Controller

Michael Trampler, Javier Arias, Ryan Kastovich, and Genaro Moore

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The objective of this project is to design an HVAC Control System with capabilities of creating schedules, creating set points for zones, obtaining accurate readings for temperature, CO2 and Humidity, and controlling the HVAC system to specific zones. Multiple remote sensor modules (RSM) will be implemented so the user will be able to control temperature and humidity in certain zones through an aesthetically pleasing interface. This system will also feature internet connectivity for the convenience of control anywhere there is internet access. The web interface will give the user the control features of an RSM, while the user is away.**

***Index Terms* — Centralized control, Data structures, Microcontrollers, Radio Transceivers, Thermal sensors, Web design, Wireless Communication**

I. Introduction

Today, there are an increasing number of households running HVAC (heating, ventilation, and air-conditioning) control systems 24/7. While many of these systems might be designed to be as efficient as possible, it does not mean that they are capable enough to accommodate the needs of the user(s) in every possible usage scenario combination. For example, not every room in a house needs to be set at the same temperature at all times, especially once everyone has gone to bed. So at night, there are usually no occupants in the kitchen, living room, or dining room which are still being cooled or heated. Then, a multi-zone system was introduced to help fulfill the extra needs of consumers. With this new innovative system, users were given the ability to dictate individual temperatures to different "zones," whether they be bedrooms and living rooms, or different floors of an office building. The user could control the HVAC to cool and/or heat only the room's occupied, turning off the zones vacant through installed dampeners to control air flow. To give an example let's say there are two zones for an HVAC system, if one zone is vacant, then the user could turn one zone off directing all the air flow to the occupied zone which will then be cooled or heated faster. So with this system installed, power consumption will decrease which results in a cut in energy costs.

HVAC systems are designed to maintain a desired temperature set point. Unfortunately this system creates a large temperature gradient between the inside of the domicile and the natural weather. As per thermodynamics the larger the temperature gradient between two areas the quicker thermal energy transfers through the substrate which separates them. For example in the summer, the outside temperature can hit +90 degrees Fahrenheit while the set point for most HVAC systems will usually be between 65 and 80 degrees. This causes a large temperature gradient and reduces the effectiveness of the insulation provided by walls and purpose built insulation material. If the temperature gradient were to be reduced to a negligible value then thermal energy would stop flowing into the domicile. Our system's main goal is to reduce the temperature gradient between the interior of the domicile and the exterior during the hottest part of the day. This will greatly reduce the amount of time the heat pump's compressor will run which in turn will greatly reduce the energy it consumes. It is generally accepted that the heat pump is one of the largest energy draws in a domicile; therefore reducing its energy draw should be one of the easiest ways to reduce energy waste.

To reinforce this projects energy saving applications, here's another example of how this system will help. On a regular day in typical households worldwide, people have air conditioning units that run throughout the day while no one is home. If consideration is taken on how much energy is being wasted on a vacant house, the reality is that on a yearly basis this amount is astronomical. The advantage of the HVAC system would be the ability to shutdown the main system when no one is home. It would accomplish this task with a schedule created by the user to set only when they are not home. If by chance the user isn't home, the system could be shut off and start back up when they return. This in itself would save tons of energy and would lower the cost of running a system in a consumer's home/zone.

This system will be designed such that a consumer will be able to utilize the multi-zone controllers all while keeping the power consumption low. Although the power consumption will be low, there won't be any drop off in precision levels, customizability, or aesthetics. Multiple remote sensor modules (RSM) will be implemented so the user will be able to control temperature and humidity in certain zones through an aesthetically pleasing interface. The web app will come with preset modes with which a user can employ to run throughout the day to further

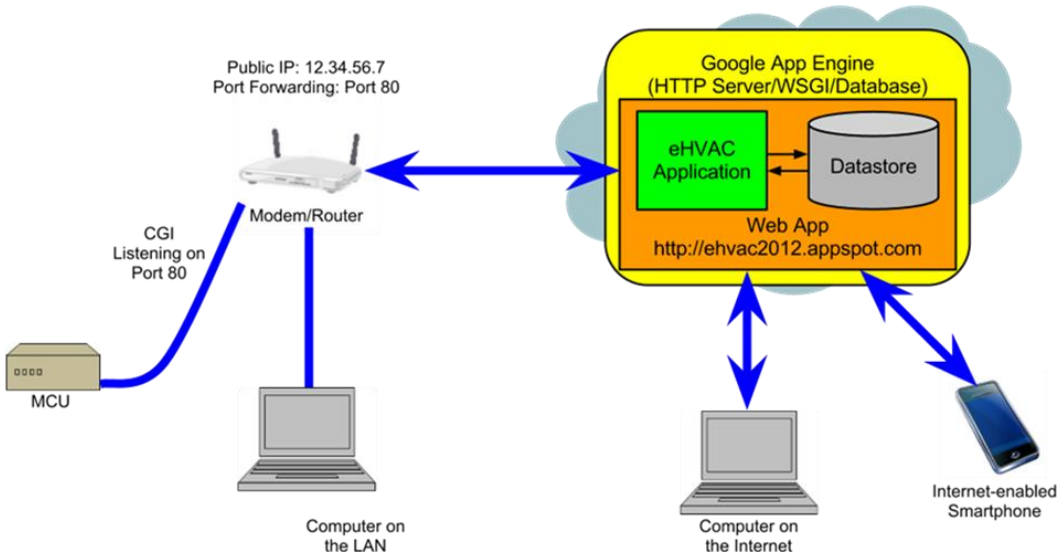

Fig. 1: End to End Connectivity of the HVAC System

decrease power consumption. But if those preset modes do not adequately meet the requirements of the user, he/she will be able to program the RSM to meet his/her own needs. This system will also feature internet connectivity for the convenience of control anywhere there's internet access. The web interface will give the user the control features of an RSM, while the user is away. An illustration of the HVAC system can be seen in Figure 1.

II. System Components

The eHVAC Control System is comprised of many components working together to carry out the total functionality of an HVAC system with little worry by the user of its accuracy and ability to perform.

A. Main Control Unit (MCU)

The Main Control Unit will be responsible for controlling the various components of an HVAC system such as the heat pump, the fan or air handler, and all of the dampers throughout duct work. It will also be responsible for bridging the gap between the web application and the Remote Sensor Modules. TI's LM3S8962 Stellaris® ARM® Cortex™-M was chosen for the MCU and was divided into two parts: hardware and software.

i. Hardware

The MCU will communicate to the RSM's via a wireless module on the Main PCB. It will also control the supply voltage to the physical components of the HVAC system which requires 24VAC for operation according to industry standards. A standard heat pump consists of a compressor, reverse valve, and emergency heating coils and sits outside the house. The air handler or fan is usually somewhere inside the house and circulates the air throughout the home. The MCU will also control up to 8 normally-open dampers which splits a household into multiple zones. If a particular zone requests for cool air while another zone does not need the air, the zone that does not require the air will have its damper closed therefore directing all the cool air to the zone that requested it. The MCU will also have Ethernet connectivity for communication to and from the web application.

The MCU will drive the plant portion of the system via two 74HC595 8-bit shift registers which will then connect to 16 different MAC97 Triacs. The shift registers will be daisy chained together so that only three inputs are required: a data input, a clock, and a latch which can be seen in figure 2 on the next page. The triacs will be connected to a 24VAC supply voltage which will drive each component. The MCU will send two hex values to shift out where each bit is responsible for a single component. So if a components' respective bit goes high, then that component will turn on via a triggering of the triac it is connected to. LED's simulate each component turning on and turning off. Since the dampers are normally-open by default, their respective LEDs will turn on to simulate the damper closing

There will also be a MSP430G2553 on the main PCB which the Stellaris will communicate with using a UART connection. This MSP430 will be connected to a

CC1101 wireless transceiver via a SPI connection. This is the wireless module that will communicate with the RSMs which also have a wireless module.

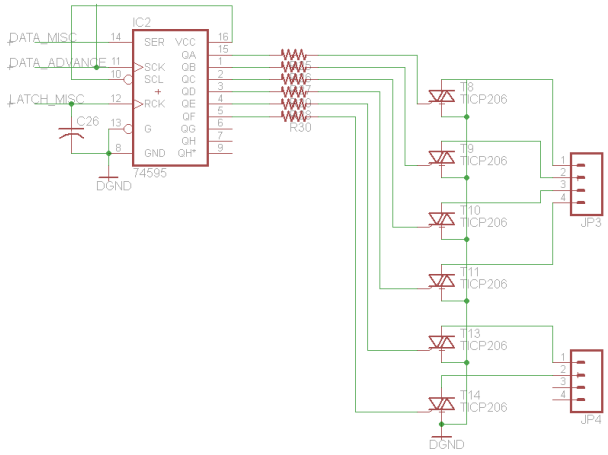

Fig. 2: Shift Register Schematic

ii. Software

The Stellaris chip has plenty of built in software functions that are advantageous to use. There are many UART functions for use including receiving and transmitting functions [1]. These functions will help tremendously with parsing through the data received either from the RSM or the web application and will be coded using the C programming language. The RSM will only be able to send data if the MCU requests it. The request to poll the RSM will first be sent from the web application. So the MCU software will be able to respond to a series of commands from the web application, execute the corresponding command, and send confirmation back to the web application.

Each command sent from the web application will have identifier values that correspond to one of the RSMs or the plant. Once this id is deciphered, the command will then be executed, whether it is requesting data from the RSMs, sending new data to the RSMs, updating the state of the plant either from a scheduled event or fluctuation in temperature readings.

If the web application requests data from the RSMs, the data sent via the wireless module will first be stored and parsed on the Stellaris chip, then a string of data will be prepared and sent immediately to the web application. The web application sends requests and commands with the Stellaris through CGI calls, but the Stellaris will prepare the data to send back in JSON string format for the web application to decipher in Python. This implementation was chosen because there was source code for JSON parsers available for both the web application and the Stellaris but the parser was too large to fit onto the Stellaris so the CGI implementation was chosen.

B. Remote Sensor Modules (RSM)

The Remote Sensor Module (RSM) acts as the main data input for the HVAC controller. The RSM has several responsibilities; it must gather data concerning the environment in which the RSM is placed, it must act as an input for the HVAC controller, and it must display real time data for the user.

To accomplish these goals the RSM has a 1.8" TFT color display, several momentary pushbuttons, a Telaire T6004 carbon dioxide sensor, Honeywell's HIH-6130 humidity and temperature hybrid sensor, and to control it all, a MSP430G2553 MCU, which communicates with the HVAC MCU via the wireless module. The T6004 is a Non-Dispersive Inferred sensor (NDIR). NDIR sensors are typified by very great accuracy, low power, and high cost, and the T6004 is all of these. The T6004 carbon dioxide sensor was chosen because of its incredibly high accuracy (40ppm CO2) and because it is no longer in production, which means it is available for a small fraction of its retail price (100US) [2]. The T6004 draws 15mA at 5VDC when sampling (every 2 seconds) and under 1mA when not. Unfortunately the T6004 has a custom serial peripheral interface (SPI) which means a custom SPI protocol had to be written to interface with this sensor. This protocol requires the use of an ACK acknowledge, which causes the RSM MCU to pause communications until the CO2 sensor is ready to receive the next byte of information. The protocol also requires that the sensor operate at a frequency of at least 1 MHz transmission. yet it must only send one byte at a time. The protocol requires that multiple and variable number of bytes be sent during one transmission session. When receiving data from the T6004, the protocol requires that differing lengths of messages be received. In the first 3 bytes the T6004 will send the length of the message being sent and the receiver must receive only that number of messages. If the RSM MCU sends an unexpected value, or receives in an improper manner the T6004 will lock up and reset. If the T6004 resets it will be unavailable for one minute while it warms up and does internal checks.

Honeywell's HIH-6130 is a very accurate temperature and humidity sensor designed specifically for HVAC systems. The HIH-6130 has a temperature accuracy of ±0.25ºC and a humidity accuracy of ±4% [3]. It offers custom firmware which compensates for sensor drift over time which is built directly into the sensor, eliminating the need for external compensation. The HIH-6130 communicates using a custom SPI protocol, and a set of custom SPI functions was written to interface with it. This

protocol requires that the RSM MCU receives and stores four bytes at once. There is no minimum frequency for the HIH-6130 communication protocol, and the HIH-6130 will easily handle improper communications by simply not returning anything and waiting for the RSM MCU to perform another measurement request and read the updated measurements. The HIH-6130 provides status information in addition to measurement data. The custom protocol will take this data into account and re-poll the HIH-6130 if necessary to receive updated and accurate information.

Figure 3 depicts the daughter boards used to interface with the HIH-6130. This board contains all the support components requires to power and operate the sensor. The daughter board provides easy access to power, ground, SPI clock, SPI chip-select and SPI master-in-slave-out (MOSI). This chip has no input, only using the chip-select and clock pins to drive it.

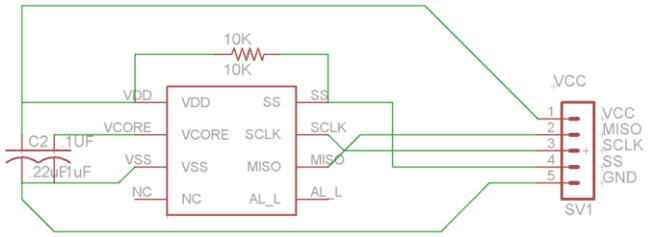

Fig. 3: HIH-6130 Daughter Board

The 1.8"TFT display is a low cost display which communicates over a traditional SPI bus, but the hardware SPI bus of the RSM MCU is used for UART. Thus the custom protocol written for the HIH-6130 was adapted for use with this display.

The MSP430G2553 was chosen to drive the RSM because of its very low cost, its ease of prototyping, and its large flash memory (16Kb). The RSM MCU communicates with all of the sensors, the display, and the wireless module. The RSM is in low power mode 0 (LPM0) when not processing an interrupt. Figure 4 describes the program response to an interrupt. The interrupt can be a request for a status update which originates in the HVAC MCU, a measurement request which originates in the HVAC MCU, or a timer interrupt which originates in the RSM MCU. regardless of the interrupt type, once the interrupt is has been fully processes the RSM MCU returns to LPM0.

C. Wireless Module

The Wireless Module is the communication bridge between the RSMs and the HVAC MCU. The module

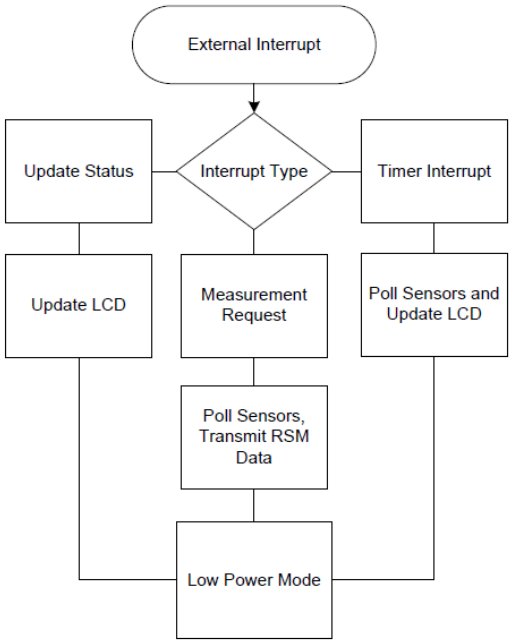

Fig. 4: Flowchart for Program Response to an Interrupt

consists of a CC110L transceiver, and a MSP430G2553. The CC110L operates at ~900MHz. The output power of the CC110L can be modified such that the range will vary between ~2 and 50 meters, which allows the user to reduce power consumption if the range requirements are small. The CC110L communicates via a traditional SPI bus, and requires a few external general purpose input output (GPIO) pins for interrupts. The MSP430G2553 acts as a controller and buffer for the CC110L. It initializes the CC110L, handles its interrupts, and communicates to the host hardware(either the HVAC MCU or a RSM) via UART. The wireless module will take an input from the host hardware via UART and then broadcast that message to any listening wireless module. In the same manner, when the module receives a wireless transmission, it will transmit the message via UART to the host hardware.

Figure 5 shows the schematic view of the Wireless Module. The MSP430G2553 drives the module. SV3 and SV4 interfaces with the support hardware, and contains power and ground, a reset pin which needs to be help high, and UART TX and RX pins. These five lines connect directly to the MSP430G2553 without any further abstraction. The two other headers SV1 and SV2, interface with the CC110L, and they include power, ground, SPI connections (clock, chip-select, master-out-slave-in(MOSI), MISO), and two GPIOs used for interrupts. One line allows the CC110L to drive the MSP430G2553 into a receive interrupt, the other allows

the MSP430G2553 to drive the CC110L into an internal transmit interrupt.

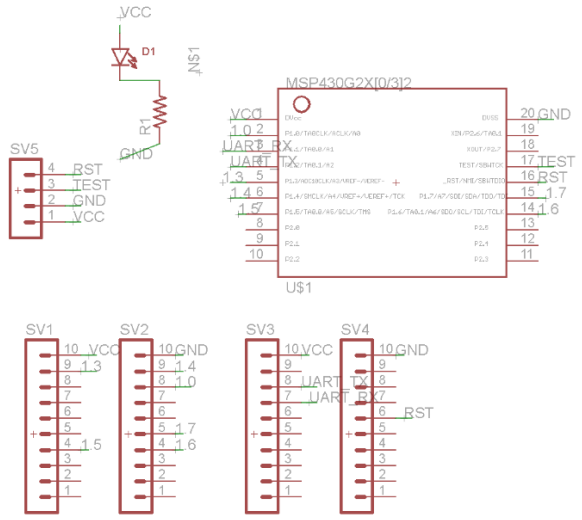

Fig. 5: Wireless Module Schematic

Figure 6 describes the interrupt handlers for the wireless Module. If the interrupt is a byte which needs to be transmitted, the module will switch the CC110L into transmit mode (the amplifier will be in line to the antenna, while the receiver will be switched out of line) and the radio will transmit the byte after which, the radio will go into listen mode (the amplifier is switched out of line, and the receiver is switched inline) and the MSP430G2553 will go into LMP0. If the interrupt is a receive type the radio will store the received byte in its RX buffer and the MSP430G2553 will read from this buffer. Once the byte is read, the MSP430 will transmit it to the host hardware via the UART lines and then go into LPM0.

D. Web Application

The web application is comprised of three main subsections: the User Interface, the Google App Engine and MCU Connectivity.

i. User Interface

The User Interface of the eHVAC system refers to the web application the user will use to control all aspects of the system. The web application was created using jQuery Mobile, which is a Javascript library made mostly for use in mobile devices such as smartphones and tablets. This will allow the user to control the system from almost any device with an internet connection and a browser with a clean looking interface. Also, the web application was coded using HTML5 and CSS3 styling for the web pages. The jinja2 templating engine will also be used in python handlers which will be discussed later. This templating engine takes care of preparing the HTML responses with the appropriate information for the user.

The system has access control in place, so a user needs the appropriate login information to gain access. The access control system only allows users control of zones and settings the administrator has granted access to. Along with being secure, the web applications allows the creation of new zones, schedules for each zone, and customize settings for set point, fan modes and system mode.

The web application has a dropdown menu and has options for settings that are mainly used for the Administrator. These settings include giving the Administrator the ability to create other users, edit users zone accessibilities, change users passwords and allow the Administrator to change the HVAC configuration settings which enables this HVAC controller to be compatible with any existing HVAC system. One asset of our web application that was quite important to implement was the readings portion. The user will be able to choose a specific zone in their system and will be able to see the most recent readings for temperature, CO2 and humidity in a very well

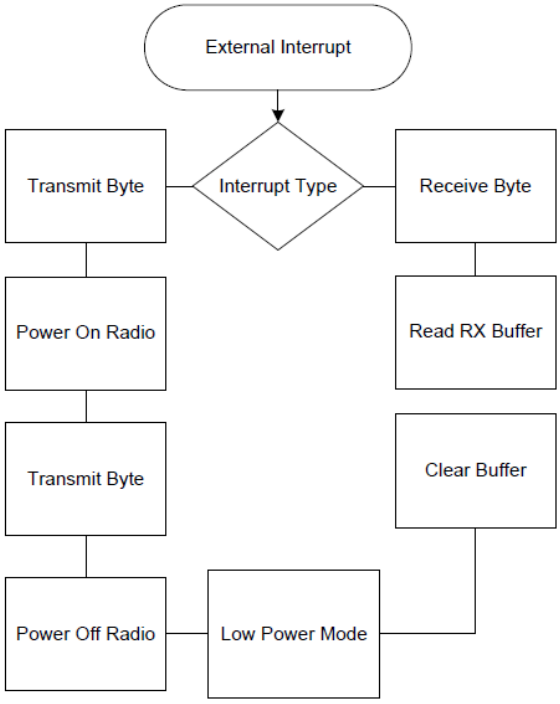

Fig. 6: Flowchart for Interrupt Handlers for the Wireless Module

designed graph structure. A depiction of the User Interface can be seen in Figure 7.

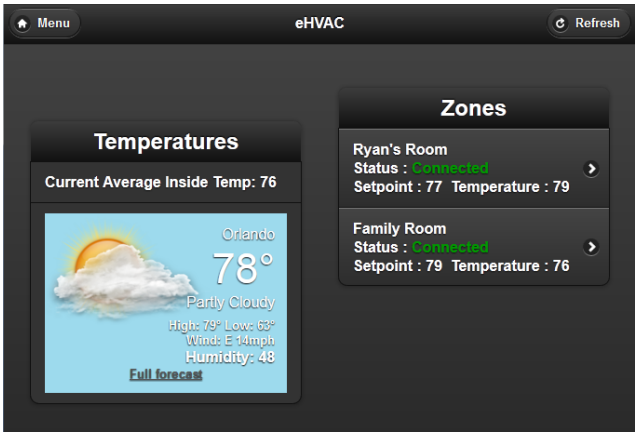

Fig. 7: GUI Design

ii. Google App Engine

The Google App Engine (GAE) is a large asset to the working structure of the eHVAC system. It provides a development environment with its own datastore, web server, and web related services. It allows for development of applications in Go, Python, or Java. The datastore has an object-oriented nature which allowed for modeling of data using classes. The GAE datastore and python capabilities allow for rapid execution and information processing in the application with minimal complications. Webapp2, which is the lightweight framework built-in to the Google App Engine, allows for a clean and simple to understand application architecture design, with exception handling.

The webapp2 application was coded with a Web Server Gateway Interface (WSGI) setup for routing requests. Python handlers were created to handle the different activities in the application and effectively creating a sandbox for the different actions and operations in the application. When a page is requested the WSGI router dispatches the appropriate handler. The most current information will be displayed to the user, based on their access level. The templating engine discussed earlier in the User Interface section was Jinja2. Jinja2 is a templating system that accounts for dynamically generated portions of HTML and will also allow for embedding special placeholders in the HTML files to indicate where the generated content should be placed.

One of the biggest portions of the eHVAC control system is the database. After much research, it was decided that the Google App Engine Datastore would be used for the database. The Datastore is a quite simplistic form of database which consists of creating models for our database from a python file. The Datastore "is a schemaless object datastore providing robust, scalable storage for your web application" [4]. The Datastore has a good hierarchical system which allows for creating classes that share the same parent and will allow for queries in the database to extract only specific data needed to display to the user when requested. The transactions that will occur from the web application and the MCU will make the database a large proponent in getting all the aspects of the project working. The database will take requests from the users via the web application if they change any settings. The values that will change will be placed into the database and if needed will then be sent to the MCU to change any settings that the microcontroller's need to operate (fan mode, system mode, and set point). The flowchart in Figure 8 shows an example of how the system handles a request to change the Fan mode in a zone.

The main models in the database include: Plant, Zone, User and Schedule. The Plant and zone models are very important because they house all of the needed components for each plant and zone in the system. If a user was to add a new zone to their system, they would need to use the Zone model to create this and have it place all of its values in the corresponding Plant. The User model controls all the properties for when a user has been created to giving a user access to zones and setting administrative access. Finally, the Schedule model is needed for the user to be able to create special times for when their system will be in the mode of their decision along with choosing a set point. The Schedule is also very important when using the Cron handlers needed to have the system pull the most recent set schedule from the database and to have the system run it at the correct time.

iii. MCU Connectivity

Communications with the MCU is done via a combination of web technologies and standards. The application in the Google App Engine contains a Cron daemon which executes a python script called pollSystem at a specific frequency of once every two (2) minutes. The flow of this script is represented in Figure 9. The script first polls the MCU by executing a method called getSystem. This method sends a command via the MCUs CGI (Common Gateway Interface) requesting all plant and RSM data. The expected response from the MCU is a JSON string with the most up-to-date information about the plant and RSMs. After obtaining the JSON response the pollSystem method parses through the RSM information in the JSON, and updates the zone readings accordingly. Once all readings have been parsed and processed the method sends a command over CGI to the MCU with the newest zone settings from the web

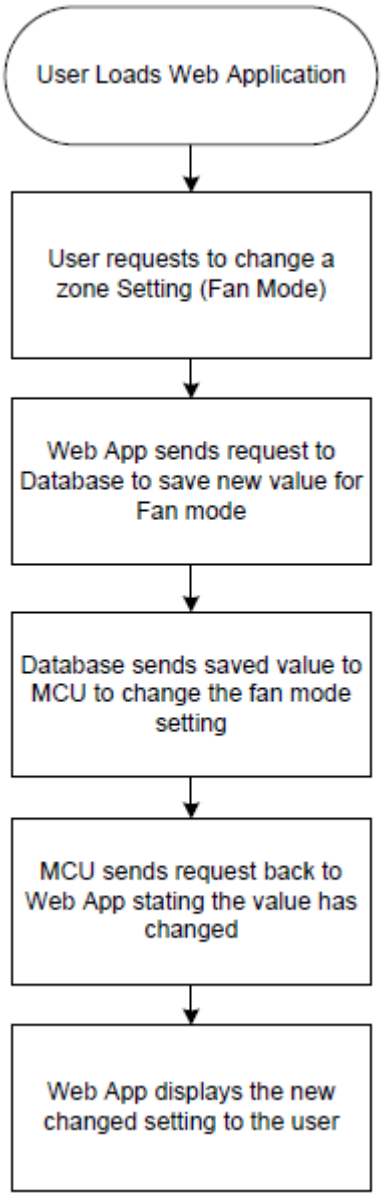

Fig. 8: Flowchart for Fan Mode Change Request in the System

application, ensuring that all RSMs are also up-to-date with the desired set point, system mode, and fan mode for each zone. Here is where the scheduler information comes into play.

The script will try and query the Datastore for the schedule that should be running (if applicable) in each zone. This schedule is dependent on the day of the week, start time and whether or not it is enabled. This start time cannot be greater than the current time so as to avoid executing schedules ahead of time. If there is no schedule to be executed, pollSystem will default to the main zone settings for each zone. The expected response for each of these CGI commands is a simple acknowledgement in the form of a JSON string with the same settings that were sent in the command. This allows pollSystem to verify that the MCU received the correct information. Last but not least the script will prepare the appropriate plant output according to current zone settings and readings. This output is represented by two 2-digit hex numbers. One number contains the plant operation settings, while the other contains the settings for the system dampers for controlling air-flow. Once these two numbers are ready, pollSystem will send them to the MCU via another CGI call. The expected response of the MCU to the CGI call is a JSON string with the information it received. If the response is valid then it will update the plant information in the Datastore with the new plant output values. In case the response is not as expected it will finish the method. Regardless of response, the Cron daemon will execute the pollSystem method in another 120 seconds.

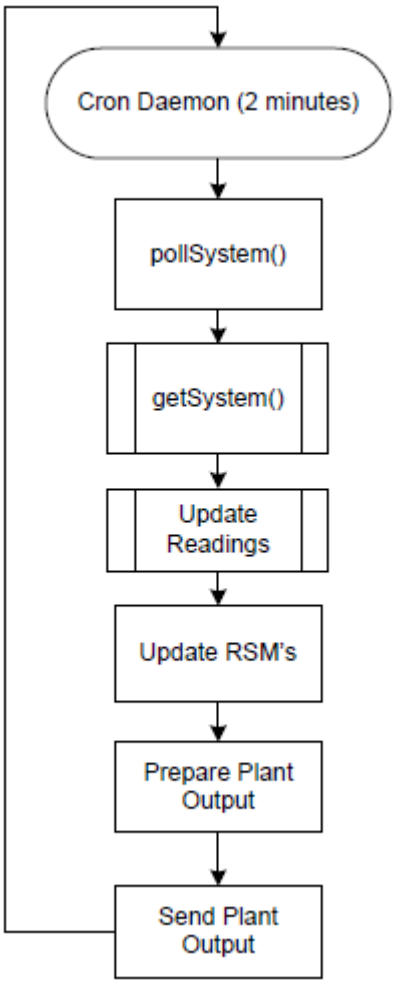

Fig. 9: Flowchart for a Cron-Handled Request in the System

III. Conclusion

The eHVAC project was a good learning experience for all members of the group. It not only tested the knowledge and skills we had prior to taking Senior Design, it also utilized our patience and well-being to put forth the effort needed to succeed. Many issues lingered throughout testing of the project including the intermittent Stellaris microcontroller and its lack of help and support. We felt as though we accomplished much more because we managed to pull through the hard issues and created our own work-a-rounds for completing this project. We hope to work with this project in the future and possibly expand on it.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Samuel Richie, Dr. Elena Flitsiyan, Dr. Zakhia Abichar, Dr. Saeed Lotfifard, Dr. Mainak Chatterje and the rest of the UCF EE/CPE Department.

BIOGRAPHY

Michael Trampler will be graduating UCF in the Fall of 2012 with his B.S. in Electrical Engineering. He will pursue his master's and Phd in Electromagnetics and Optics with Dr. Gong. After earning his PhD, Michael intends on pursuing a job in the field of Microwave Engineering.

Javier Arias will be graduating from the University of Central Florida with a B.S in Electrical Engineering. After graduation, he will look into the fields of communications, power electronics and power engineering. Javier intends on pursuing a master's degree in Power System's or Optic Engineering.

Ryan Kastovich will be graduating the University of Central Florida with his BS in Electrical Engineering in the Fall of 2012. He will continue his work at his job at LiveTV as a fleet engineering intern until he has a full time position. Ryan plans to pursue a career in the aviation industry working as an electrical engineer. He plans to further his career with a Master's degree in Digital Signal Processing.

Genaro Moore is an electrical engineering major graduating this December. After graduation he will accept a full-time offer from either LCEC in Fort Myers or Crunchy Logistics here in Orlando. Genaro intends on pursuing a master's degree in Business Administration and a Bachelor's in music later on in his career.

References

[1] Stellaris Peripheral Driver Library. Dallas, Texas: Texas Instruments Incorporated, 5 Sept. 2012. PDF.

[2] Telaire 6004 CO2 Module. General Electric, 2006.. PDF.

[3] Honeywell HumidIcon Digital Humidity/Temperature Sensors: HIH6130/6131 and HIH6120/6121 Series. Golden Valley, Minnesota: Honeywell, July 2012. PDF.

[4] "Datastore Overview- Google App Engine." Google Developers. N.p., n.d. Web. 21 Apr. 2012.< https://developers.google.com/appengine/docs/python/datastore/overview/>.