SELF ORGANIZING AIR VENT SYSTEM

by
Kyle Schwab

A project
submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Boise State University

April 2016
Project Title: Self Organizing Air Vent System

Date of Final Oral Examination: April 2016

The following individuals read and discussed the project submitted by student Kyle Schwab, and they evaluated the presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Tim Andersen, Ph.D. Chair, Supervisory Committee
Jidong Xaio, Ph.D. Member, Supervisory Committee
John Gardner, Ph.D. Member, Supervisory Committee

The final reading approval of the project was granted by Tim Andersen, Ph.D., Chair of the Supervisory Committee. The project was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.
Dedicated to Annie and Mya.
ACKNOWLEDGMENTS

A very big thank you to my family members for providing their support, time, and encouragement with helping me accomplish this endeavor. Thank you to my professors for their wise words, their counsel, and for providing the inspiration to continue pushing my boundaries. What an experience with such great individuals, I cannot express my appreciation enough.

I would also be remiss if I did not mention my employer, Micron Technology, for providing the finances to make this possible. Finally, thank you to my fellow co-workers for being flexible and supportive while pursuing this achievement.
ABSTRACT

The average residential household is controlled by a single-zone Heating, Ventilation, and Air Conditioning (HVAC) system. A single-zone refers to a configuration in which all rooms receive conditioned air during HVAC operation. These single-zone HVAC systems usually consist of a temperature sensor integrated into a centralized thermostat controller installed in the home. These centralized thermostat controllers can only monitor and detect the localized temperature of the immediate surrounding area, thus, not representative of all areas within the home. Ideally, the temperature recorded and set at the centralized thermostat would be reflected in all areas throughout a residence. However, most rooms do not receive the correct air conditioning to maintain a desired temperature; this results in uncomfortably hot or cold areas within a residence.

There are limited methods available to consumers for correcting these temperature imbalances. The first option requires an expensive retrofit of the HVAC system; this allows for the installation of a multi-zone system, improving upon a single-zone system’s inadequacies. However, a retrofit consumes a customer’s time and intrudes upon their home.

The second option is similar to the Activent system, a lower cost alternative to expensive retrofits which attempts to correct room-to-room temperature imbalances with automated vents. While cost effective, the Activent system does not allow for zone to zone communication, often leading to temperature variations throughout a household.
The Self Organizing Air Vent (SAVE) system provides an affordable solution that is cost effective, easy to install, and resolves the temperature variations seen in single-zone systems. The SAVE system is a wireless ad-hoc mesh network of automated vent registers and zone controllers, that cooperatively work together to evenly distribute the temperatures across a home. This report covers the prototype electrical boards and control firmware needed to test and validate the SAVE system. Simple testing chambers were used for experimentation and collection of results.
# TABLE OF CONTENTS

ABSTRACT ................................................................. vi

LIST OF FIGURES ....................................................... ix

LIST OF ABBREVIATIONS ............................................... x

1 Introduction ............................................................ 1

1.1 The Central Thermostat Problem ................................. 1

1.2 Current Solutions .................................................. 2

2 The SAVE System .................................................... 5

2.1 Overview ........................................................... 5

2.2 Zone Controllers .................................................. 6

2.3 Automated Vent Registers ........................................ 6

3 Prototype System and Implementation ............................. 8

3.1 Project Statement and Deliverables .............................. 8

3.2 Hardware Implementation ......................................... 9

3.2.1 Board Revision 1 ............................................... 12

3.2.2 Board Revision 2 ............................................... 14

3.2.3 Board Revision 3 ............................................... 15

3.3 Firmware Implementation ......................................... 17

3.3.1 Firmware Architecture ....................................... 18
LIST OF FIGURES

1.1 Activent “Micro” Switch Pairing Implementation .......................... 4

3.1 Prototype Vent (Mechanical Rev 1, Electrical Board Rev 2) ............ 11
3.2 Board Revision 1 ........................................................................ 13
3.3 Board Revision 2 ........................................................................ 15
3.4 Board Revision 3 ........................................................................ 17
3.5 Basic Firmware Architecture ...................................................... 19
3.6 The Event Task Data Structure ..................................................... 23
3.7 Event Task Data Flow Diagram .................................................... 24
3.8 14 Byte Radio Packet Structure ................................................... 30
3.9 Example of SAVE Mesh Network Topology ................................... 33
3.10 Automated Vent Register and Save Controller Interaction .......... 35
3.11 Device Pairing Sequence Diagram .............................................. 38
3.12 Temperature Balancing Example ................................................ 45

4.1 Temperature Balancing Testing Chamber ...................................... 48
4.2 HVAC System State Testing Chamber .......................................... 48
4.3 Temperature Variation with SAVE System Disabled ..................... 49
4.4 Temperature Variation with SAVE System Enabled ...................... 50

A.1 The Gold Phoenix PCB PO ......................................................... 56
LIST OF ABBREVIATIONS

API – Application Program Interface
APL – Application Abstraction Layer
ASF – Atmel Studio Framework
AVR – Advanced Virtual RISC
BOM – Bill of Materials
CAV – Constant Air Volume
CRC – Cyclic Redundancy Check
DART – Discharge Air Regulation Technique
FCFS – First Come First Serve
FIFO – First in, First out
GPIO – General Purpose Input/Output
HVAC – Heating, Ventilation and Air Conditioning
IDE – Integrated Development Environment
ISM – Industrial, Scientific and Medical (radio bands)
ISR – Interrupt Service Routine
MCU – Microcontroller
NLOS – Non-Line-of-Sight

xi
**PAL** – Peripheral Abstraction Layer

**PCB** – Printed Circuit Board

**PMIC** – Programmable Multi-level Interrupt Controller

**PO** – Purchase Order

**RF** – Radio Frequency

**RISC** – Reduced Instruction Set Computer

**RTC** – Real Time Clock

**SAVE** – Self-Organizing Air Vent

**SPI** – Serial Peripheral Interface

**USART** – Universal Synchronous/Asynchronous Receiver/Transmitter

**USB** – Universal Serial Bus

**VAV** – Variable Air Volume

**WPS** – Wi-Fi Protected Setup
CHAPTER 1

INTRODUCTION

1.1 The Central Thermostat Problem

Most residential homes are outfitted with a Heating, Ventilation, and Air Conditioning (HVAC) unit that controls the temperature of a home with conditioned air. Generally these HVAC systems are installed in a single-zone configuration, meaning that all of the rooms receive conditioned air during HVAC operation. These single zone HVAC systems usually only consist of a single temperature sensor that is embedded within a centralized thermostat controller. While recent advancements have been pushing more intelligence into these centralized controllers, they all still suffer from a critical issue: the inability of detecting and accounting for the temperature variations throughout a home.

There are several reasons why some rooms within a house heat or cool at different rates. The thermal characteristics of the region, the occupancy count of individuals, the HVAC air duct installation quality, and a room’s location within a household are all examples of factors that contribute to temperature variations. These centralized thermostat controllers can only read the localized temperature for the surrounding area. Thus, the temperature reading collected at the thermostat oftentimes manifests in peripheral rooms as being uncomfortably warm or cold since it is unable to detect and correct the variation. Specifically, these centralized thermostats leads to zoni-
ification, which are large temperature differences between rooms on the same floor, and stratification, which are large temperature differences between rooms on different floors.

1.2 Current Solutions

The current primary solution used for solving these room to room temperature variations is to employ a multi-zone HVAC system. A multi-zone HVAC system allows for a house to be partitioned into several zones to more effectively heat and cool different areas. Typically multi-zone systems are configured in homes with more than one floor, where each floor consists of its own heating and cooling zone. These multi-zone systems offer better temperature balancing across a household as well as an increase in energy efficiency when compared to traditional single-zone HVAC systems. However, these multi-zone systems still tend to suffer from room to room temperature imbalances within a single-zone. In addition, if a home is not pre-installed with a multi-zone HVAC system, the cost to retrofit and install one is prohibitive for the average consumer.

More advanced multi-zone HVAC systems are available in the commercial sector that utilize wireless capabilities and other control mechanisms for enhanced system performance. The Discharge Air Regulation Technique (DART) [4] incorporates a mesh network of temperature sensors to control HVAC fan speed, effectively changing a Constant Air Volume (CAV) system into a Variable Air Volume (VAV) system. Millennial Net [9], and similarly Siemens APOGEE [13], provide enhanced building control and measurements through the use of wireless temperature nodes. However, these systems are not offered to consumers and are employed in corporate or industrial
settings. In addition, they still require retrofitting the existing system.

A more consumer friendly solution is the Activent [1], which easily installs into an existing HVAC system without the need for retrofitting. It uses wireless communication in conjunction with automated vent registers to open and close air vent louvers. In practice however, the Activent suffers from technical limitations that decrease its actual effectiveness. The first limitation of the Activent system is the inability for zones to communicate with one another. This essentially reduces the Activent to several smaller single-zone systems which again leads to temperature imbalances between the zones. Another limitation is the Activent’s manual pairing process, shown in Figure 1.1, which requires a user to configure the system through a series of micro-switches on the devices. This manual pairing process decreases the ease in which a user can quickly reconfigure their system, as well as making the initial setup process more cumbersome and non-intuitive.

Ideally, a preferred solution to the centralized thermostat issue would be a multi-zone system that accounts for and balances the temperature variations across a home. It would also need to be low cost, easy to install and configure, and not require retrofitting the existing HVAC system. These were the defining characteristics that drove the implementation of the SAVE system.
Figure 1.1: Activent “Micro” Switch Pairing Implementation
CHAPTER 2

THE SAVE SYSTEM

2.1 Overview

The Self Organizing Air Vent (SAVE) system is a solution intended to solve the inadequacies of centralized thermostat controllers and multi-zoned HVAC systems. The goal of the SAVE project is to further improve room to room temperature balancing while still being cost effective and easy to install by a homeowner. This means that the SAVE system will have to be capable of being installed with minimal to no impact on an existing HVAC system.

The SAVE system is composed of automated vent registers and zone controllers that wirelessly communicate to monitor and control HVAC airflow output. The SAVE system effectively works by redirecting airflow (by closing or opening air vent registers) among the participating zones to actively balance the temperatures between them. An example of one such scenario could be during an HVAC heating cycle. Imagine a room that is hotter than the desired temperature. The warm air that is entering that room is wasted, and could be better used (or redirected) to a room that is below the target temperature. Using this basic idea, the system redirects and distributes the airflow across the rooms to balance the temperature differences between them. Over time the SAVE system is able to influence the various temperatures across the different rooms to maintain the same temperature.
The goal of the system is to provide an affordable alternative to current multi-zone systems. A house can be divided into zones by simply pairing one or more vent registers in a room or area to a particular zone controller. These zone controllers then can work together to ensure that the temperatures seen within the zones are balanced across all of the zones.

2.2 Zone Controllers

Zone controllers are responsible for controlling some designated region and monitoring a localized temperature. A region could be defined as a single room, or multiple rooms depending upon the desired zone configuration. However, the temperature of the zone is determined by the localized temperature as seen by the zone controller. A zone controller manages its region through the use of automated vent registers that have been paired with the controller itself. These automated vent registers control the louvers that either allow or inhibit the passage of air flow from the HVAC ducts into the region. Each zone controller is then responsible for coordinating with other zone controllers to actively balance the temperature distribution seen across the zones to achieve temperature equilibrium.

2.3 Automated Vent Registers

Automated vent registers are responsible for opening and closing the vent louvers of the HVAC ducts, in addition to detecting the state of the HVAC (heating vs cooling, on vs off). The vent registers are designed to pair with a zone controller which dictates the opened or closed state of the automated vent louvers depending upon the states of other zones. Automated vent registers may also be used to monitor the health
of the HVAC system. For example, based on pressure readings, the system may be able to detect when the HVAC filter needs to be replaced or if other maintenance is required.
CHAPTER 3

PROTOYPE SYSTEM AND IMPLEMENTATION

3.1 Project Statement and Deliverables

The graduate project statement is to:

Successfully develop both the necessary firmware and hardware to enable control of an automated ventilation network capable of evenly distributing temperatures across multiple zones.

The project focused on four key deliverables for the enablement of the SAVE prototype system that relate to the Computer Science field. Specifically, the four deliverables were to:

1. Develop the electrical boards to manage the zone controllers and automated vent registers.

2. Develop the control firmware to form an ad-hoc wireless mesh ventilation control network.

3. Develop firmware to enable an automated pairing process between zone controllers and automated vent registers.
4. Develop an algorithm to reliably detect the HVAC system state (active or inactive) through the use of a low cost pressure temperature sensor; the Freescale MPL3115A2.

### 3.2 Hardware Implementation

The electrical board development focused on the process of developing a custom prototype board used for testing the SAVE system. The prototype board underwent three iterations, the first two of which were larger incremental changes, and the final iteration consisting of minor adjustments for the completed implementation. One hardware design goal for the project was to incorporate and use low cost parts as much as possible. The project was successful in keeping the bill of materials (BOM) unit cost low, at about $33 for parts (including the motor), PCB manufacturing, and board assembly for 60 units. The target goal was to keep these costs under $25 and under volume production with a crossover point of over 100 units, the project met the low cost goal. A one time cost for the PCB board stencil creation was not factored into the goal calculation, however would also be negligible over volume production.

The entire SAVE electrical board BOM and Gold Phoenix PCB purchase order (PO) is included in Appendix A for reference. The three primary components of interest to the project however are:

1. The Atmel ATxmega64D4 low power 16 bit Microcontroller (MCU).

2. The Freescale MPL3115A2 pressure temperature sensor.

3. The Microchip MRF49XA 433, 868, and 915 MHz ISM Band Sub-GHz RF Transceiver.
The Atmel ATxmega64D4 is a 16 bit low power Advanced Virtual RISC (AVR) MCU with 64 KB of Flash. The ATxmega64D4 was chosen for a few reasons to be used as the MCU for the SAVE prototype board. The first was the excellent design support offered by Atmel through the Atmel Studio Framework (ASF) that provides MCU software libraries for their MCU products. The ASF is very useful as it provides a quick path for rapid evaluation, prototyping, and design for embedded applications when using their MCUs. The second reason was the ATxmega64D4 had great power characteristics in low power states, the necessary SPI/I2C/USART hardware interfaces, a Programmable Multi-level Interrupt Controller (PMIC), and two extra ports for General Purpose Input/Output (GPIO).

The selection of the Freescale MPL3115A2 was primarily driven by selecting a cost reduced solution for a pressure temperature sensor. In the case of the MPL3115A2 the reduced cost came with a tradeoff in precision, with a sensitivity of only 50 to 110 kPa. Ideally, a pressure sensor with a higher sensitivity (under 50 kPa) would have been a better fit for the application, however the costs associated were prohibitive given the cost goals. The MPL3115A2 supports programmable events and autonomous data collection for both pressure and temperature, with a FIFO depth of 32 samples over an I2C interface. A goal of the project was to determine whether firmware could reliably compensate for a low cost pressure temperature sensor with respect to detecting the HVAC system state (on vs. off, heating vs. cooling). The MPL3115A2 was used to evaluate the validity of firmware compensation for decreased pressure resolution for the SAVE system.

Another cost driven selection was the Microchip MRF49XA RF transceiver. By choosing a lower cost radio transceiver, the selection of the MRF49XA meant sacrificing features and functionality often found on higher cost transceivers. Specifically,
the lack of any radio stack or library, limited receive FIFO depth of only 2 bytes, and other more advanced wireless capabilities such as packet re-transmission and collision handling. The MRF49XA is a Sub-GHz ISM multi-band RF transceiver with a SPI communication interface. The MRF49XA is fairly bare bones and a relatively straightforward RF transceiver, providing basic RF capabilities. Again, with the low costs goals set for the project it was the responsibility of firmware to develop a robust radio stack and protocol to attempt to compensate for the limited feature set of the MRF49XA.

![Prototype Vent (Mechanical Rev 1, Electrical Board Rev 2)](image)

Figure 3.1: Prototype Vent (Mechanical Rev 1, Electrical Board Rev 2)

Since the project focused around providing a prototype testing environment for the SAVE system, a key goal was to enable easy debugging from a hardware perspective. For this purpose, the board design included all signal lines routed to headers to allow for easy debugging through the use of a logic analyzer. This proved to be an invaluable decision through the development process, as it made hardware communication debugging very easy. A Saleae Logic Pro 16 [10] was used for analyzing the board signal lines and resolved several issues that would have otherwise remained open.

The board design utilized a mini USB type B port for supplying power, with the ability through jumper selections to also be powered via a 9V battery. Power jumpers
were also included for the three primary components (MCU, pressure temperature sensor, radio transceiver) as well as the board power to allow for power consumption measurements. A servo motor was used in lieu of a DC motor for the vent louver control as it allowed for a quick prototype implementation for start/stop and positional locations of the motor. Preferably, a lower power implementation would use a DC motor in conjunction with other sensors. However, this was determined to be outside of the scope of the project as integration with the mechanical prototype would have been overly complicated. The board dimensions were also derived from the mechanical prototype implementation with a max length and width specified at 2.0” x 3.5”, and the final board implementation dimensions at 2.0” x 3.425”.

A final goal for the hardware implementation was to implement an embedded PCB antenna. This was primarily driven by wanting to ensure long range communication at a reduced cost without having to use a chip or external physical antenna. The project explored two different antenna implementations on different Sub-GHz bands. The application notes from Silicon Labs [11, 12] were used to guide the antenna design, with the first antenna implementation using the 433 MHz band on a “Big Loop” PCB antenna. The final prototype PCB antenna used the 915 MHz band “Xtapped Loop” design.

3.2.1 Board Revision 1

The first prototype board, Figure 3.2, for the SAVE system was to lay the groundwork for the second two iterations, and was intended to provide an overall electrical test of the components, signal lines, and PCB antenna implementation. The only large issue found was a critical design flaw associated with the PCB antenna. This design incorporated a 433 MHz “Big Loop” PCB antenna that was originally intended as
the wireless spectrum for communication on the SAVE system. However, two issues were uncovered with this decision. The first problem was one of implementation, were the ground plane had been filled in under the antenna disturbing the transmission characteristics of the wireless signal. The board was only able to successfully communicate for distances of up to a foot. The second issue was one of optimization, where the 433 MHz spectrum was less optimal for SAVE system’s embedded PCB antenna application. The higher frequency 915 MHz spectrum offered better performance characteristics under 1 GHz and would be used instead during the board’s second iteration.

Figure 3.2: Board Revision 1
The sensor verification and other electrical tests were successful for all signal lines including the motor. A smaller batch of these boards were ordered and hand soldered for testing.

3.2.2 Board Revision 2

The second iteration of the prototype board, Figure 3.3, was focused on optimizing and fixing the PCB antenna implementation. The redesigned antenna required a significant amount of rework for the board layout. First, the antenna needed to be isolated from both the power and ground planes to reduce the transmission interference and wireless issues that were seen with the first revision. This meant moving the voltage regulators and associated passives to the other side of the board while compressing the spacing between existing components.

A “Xtapped Loop” embedded PCB antenna was implemented to move the SAVE system communication into the more optimal 915 MHz ISM band spectrum. The board thickness was also reduced from 0.062” to 0.024” to improve the transmission characteristics of the antenna. The wireless range was verified to be over 150 ft Non-Line-Of-Sight (NLOS) from within the HVAC air ducts. This long NLOS range provided adequate coverage for a residential household, and was well over the minimum range needed for testing. All other electrical tests and signal lines were again verified successfully. Similar to revision 1, the revision 2 boards were ordered in a smaller batch and hand soldered for testing.
3.2.3 Board Revision 3

The third and final iteration of the prototype board, Figure 3.4, featured only slight modifications, and was primarily focused on verification of board manufacturing and assembly. Revisions one and two essentially provided platforms for validating the design and ensuring the electrical characteristics of the board were satisfactory. The third board revision was entirely manufactured and assembled by Gold Phoenix PCB. All parts were directly sent to Gold Phoenix, and the SAVE BOM had to be modified to source the parts from Digi-Key for assembly. Only minor design adjustments overall
were made, with a slight reduction in the board size to better fit the mechanical vent prototypes. The boards were again electrically tested and all signal lines verified successfully.

The revision three board served as the prototype board used during the testing of the SAVE system, and successfully completed the first project deliverable. The revision three prototype provided all the essential components needed for satisfying either role of a zone controller or automated vent register. This dual role capability eliminated the need for two separate designs, cutting down on the time needed for both board design and verification.
The firmware implementation for the SAVE prototype system concentrated on delivering the overall system control and testing. The firmware had several challenges and goals related to overcoming the cost reduced hardware implementation and associated limitations. The bulk of the firmware development revolved around the architectural framework, the lightweight scheduler, and the design of the wireless ad-hoc vent mesh network. The firmware was written in C for the ATxmega64D4 MCU and took
advantage of the provided ASF libraries. Programming and debugging was done within the Atmel Studio IDE, using the Atmel JTAGICE3 debugger [2, 5]. SVN hosted by Assembla was used for version control of the SAVE system source code.

The specific hardware limitations associated with the selections of the MRF49XA and MPL3115A2 devices increased the complexity of the firmware implementation. The MRF49XA radio transceiver had limited collision handling, a small hardware RX/TX data FIFO, and no included radio stack or other functionality present on more expensive wireless transceivers. Thus, a significant amount of the firmware development was oriented around developing a basic radio stack, communication protocol, device authentication, and overall network topology. The lower cost MPL3115A2 temperature pressure sensor had less of an impact in terms of firmware compensation for hardware limitations. This was predominately because similar algorithms or approaches would be necessary even on costlier pressure sensors.

Just as the hardware implementation allowed for dual role capabilities between either being a zone controller or an automated vent register, the firmware implementation allowed for a dual role configuration as well. This allowed for quickly resetting a device via a button press combination to change the device type. This approach allowed for a tremendous amount of flexibility and cut down on costs and time for the prototype system implementation. Had the approach been taken to create two separate designs for both firmware and hardware, the system design, implementation, and verification would have been much more difficult.

3.3.1 Firmware Architecture

The firmware architecture was primarily designed around the SAVE system prototype project goals of providing a low power, portable and extensible lightweight code base.
To this purpose, the firmware implementation utilized rigid abstraction layers to help enforce the firmware to be more modular. The firmware implementation also targeted low power requirements as well to enhance battery life longevity. On average a typical residence contains 20 vent registers, so it was imperative to minimize the need for changing batteries frequently as it would become frustrating to consumers.

The firmware architecture was divided into 4 abstraction layers: The Application Abstraction Layer (APL), the Peripheral Abstraction Layer (PAL), Common, and the Atmel Studio Framework (ASF). The abstraction layers provided containers for hiding the implementation specific details associated within the layer. This was especially important for the APL and PAL layers as they constitute the bulk of the SAVE system control. This feature encapsulation through abstraction layers insulates the overall code base when rework is necessary, and limits the affected area to within the abstraction region.

Figure 3.5: Basic Firmware Architecture
The ASF comprises of the lowermost abstraction layer in the firmware architecture, sitting directly above the MCU interface. The ASF libraries provide low level routines, functions and drivers necessary for interacting with the MCU. This enabled rapid development by providing libraries for SPI, I2C, USART, timers, alarms, and other functionality that normally would take some development cycles to properly code and test. With the ASF, it was as simple as including the appropriate headers and running from there. The ASF also provided configuration template files for the system clocks, sleep states, communication ports, and others MCU settings that were tuned for the SAVE project application. Most of the configuration settings used the default ASF settings for system parameters. Importantly, the RTC prescaler was set to a resolution of 8 ms, which was found to be a nice compromise between system responsiveness and maintaining a low power mode of operation.

The “Common” abstraction layer differs slightly from the others as it was “vertically” integrated within the firmware architecture. This vertical integration is in reference to the overall data flow and module interactions with the Common abstraction layer. Common provides APIs and functionality that enables interactivity between both the PAL and APL. It also serves as an additional data conduit between the two boundaries, and provides other common services needed by both layers such as data stores or utilities (e.g. linked lists, ring buffers, etc.). This is in contrast to the other “horizontally” integrated abstraction layers that only interact with the abstraction layers above or below their own. The most significant service that Common provides to both the PAL and the APL is the lightweight event tasking system based upon a First Come First Serve (FCFS) scheduler.

The next abstraction layer, the PAL, focused on providing and grouping firmware libraries and functionality that directly interface with “peripheral” components. These
components include the radio transceiver, temperature pressure sensor, buttons, LEDs, LCDs, and any other external components interfaced by the MCU. The goal of PAL is to hide away the platform (or hardware) specific details from the application layer, making the code base more readily “portable”. The PAL provided APIs that generically described the necessary behavior needed by the application layer to perform certain operations, and served as a type of contract between the abstraction boundaries. The generic APIs were then remapped internally within the PAL to the platform specific APIs that actually execute the details of that contract.

Finally the topmost abstraction layer, the APL, consists of the overall SAVE system control and application implementation details. In many ways, the APL can be thought of as the heart of the system. The application layer represents the highest level of control, dictating the overall system state and mode of operation. The APL was responsible for gathering and aggregating sensory data from the pressure and temperature sensor readings, as well as adapting the system state depending upon the information received from other devices. Since the SAVE prototype electrical boards were designed to allow for dual role capability, meaning the same board could double as either a zone controller or automated vent register, the APL was similarly designed with dual role capabilities as well. This meant that the overall system state and operational modes differed depending upon the role configuration set for the board.

3.3.2 Lightweight Scheduler and Event Tasking System

The event tasking system provided a framework for system events to be mapped to task handler routines, in addition to providing a conduit through which an event and it’s associated data could be routed throughout the system. This allowed for an initial
event and it’s related data to be captured within a containing unit, and then directed between the various task handling routines to perform operational work upon the data as needed. The event tasking framework can best be thought of as providing a state machine infrastructure, where the task handler routines represent the various processing states. An event task therefore represents the trajectory of an event and it’s associated data through the various task handler processing states until completion.

**Event Task Data Structure**

The event task data structure, Figure 3.6, was created to encapsulate a unit of work consisting of the necessary meta information to fit within the scheduling framework. Each event task data structure was formatted as a doubly linked list entry that included pointers for any proceeding or following event tasks, in addition to including a reference to the linked list head. Other meta information included scheduler specific flags, timing information, prioritization, and the status of the event task. The event task data structure provided a very flexible and simple implementation for system state communication and processing. It simplified and reduced the need for other data structures and resource management while providing a simple interface to the state processing mechanics.
typedef struct _EventTask
{
    EventPriority priority;  /// < The event priority
    EventStatus status;      /// < The event status
    EventData data;          /// < The event associated data
    EventTimer timer;        /// < The event timer data
    EventFlags flags;        /// < The event flags
    EventPool* eventPool;    /// < The current event pool
    struct _EventTask* nextEvent; /// < The next event
    struct _EventTask* prevEvent; /// < The previous event
    void (*routine)(struct _EventTask* eventTask); /// < The event execution handle
    void (*callback)(struct _EventTask* eventTask); /// < The event callback handle
} EventTask;

Figure 3.6: The Event Task Data Structure

Event Pools

A goal for the firmware implementation was to prioritize static memory allocation
over the use of dynamic memory allocation to better observe, track, and enforce the
overall memory footprint of the firmware. To this end, the scheduler simply declared
a fixed array of event tasks that were then managed through the use of “event pools”.
An event pool was implemented as the head of a doubly linked list, allowing for
event tasks to be grouped and ordered into a collective. Internal to the scheduling
framework these event pools provided the scheduler various processing states to pool
the event tasks as they transition through the scheduler framework.

The scheduler maintains several event pools to manage the state transitions,
ordering, and event task processing for events. During system initialization the event
tasks are loaded from the fixed array into the event pools where they will continue to
be managed for the duration of program execution.
Figure 3.7: Event Task Data Flow Diagram
The first event pool is the Freed Pool that holds “unallocated” event tasks, and is the initial pool that event task are loaded into from the static array during system power on. From there the first system level action needed to schedule an event is to allocate and receive an event task handle. The Allocated Pool represents these system level calls for event task allocation before the event is scheduled.

Once the event task has been configured it can then be scheduled for either immediate execution, delay before execution, or repeatedly execute at some fixed interval. There are two groups of pools that serve as scheduling queues for event task execution. The first set of pools are the Scheduled Pools, which are a set of prioritized event pools that the scheduler pulls event tasks from for immediate execution. The second set of pools are the Timed Pools, which are pools that track some timed duration associated with the event before pushing them into the Scheduled Pools for execution. A repeated event upon completion of execution would then return to the timed pool to wait for it’s fixed duration of time before being pushed back into the Scheduled Pools again for execution.

The last set of scheduler pools are the Executing and Completed Pools. The Executing Pool only consists of one event task at any given point and time, as the system can only execute and process tasks serially. Upon completion of execution, the task may have been rescheduled to perform another operation and return back to one of the Scheduled Pools. Otherwise, the final destination of an event task when it has resolved all outstanding work is the Completed Pool. The Completed Pool can be entered from any of the scheduler pools excluding the Freed Pool. This is because at any point and time, an event task can be marked as complete. This is useful when an event task has been scheduled, but needs to be canceled before execution. Once an event task has been marked complete depending on the persistence, it will either
migrate back to the Freed Pool or the Allocated Pool and begin the whole process over again.

**Scheduler Implementation**

A lightweight scheduler was applied on top of the event tasking system as a fully cooperative\(^1\), prioritized FCFS scheduler. The scheduler manages the state transitions between event pools by providing and enforcing transitional rules. These transitional rules create the foundation from which the scheduler could best determine the event task execution order. The event task execution priority was maintained through the use of three separate Scheduled Pools of high, medium, and low priority. The scheduler would then selectively pull from the highest to lowest priority pools, ensuring that the higher priority pools were serviced first.

To further the goal of remaining low power the lightweight scheduler (and therefore the overall SAVE system) was designed to be event driven, meaning that until an event was triggered the system would remain in an idle sleep state. The vast majority of events are triggered during an Interrupt Service Routine (ISR), where an event task is scheduled to process the interrupt. This means that until data is ready from the radio, the pressure temperature sensor, or a timed alarm, the scheduler can maintain a low power state until some action is necessary. By scheduling an event task to service an interrupt, it allows the system to behave in a fully cooperative manner. This meant that each event task routine had to be mindful of the execution time needed to resolve the requesting event.

\(^1\)This is from the scheduler perspective. There is an exception however for radio receive events that is discussed in Section 3.3.3.
Event Task Routines

The event task routines provided a structured interface to operate on the contents of an event task data structure. An event task routine is defined as a function that takes a single pointer argument to an event task data structure. This allowed the application level processes to allocate event tasks and then assign routine handlers to the event task for the scheduler to execute when the task was scheduled to run. This greatly simplified the hand off mechanisms and interactions between the various system tasks. The task routines were subdivided into four categories: initialization routines, input routines, output routines, and process routines.

The initialization task routines provided different kinds of initialization procedures for the various SAVE system components. These initialization routines included things such as power on reset, soft reset, radio reset, and other initialization oriented tasks. The input task routines provided services for handling interrupts from external components such as the buttons or sensory data. Generally the input routines would simply fetch the data from the external interface into a local buffer and then schedule a processing task routine to process the data.

The processing task routines provided the “brains” of the application layer, were they operated on event data to derive system state. The processing routines essentially acted as a conduit between the input and output routines, where device output was dictated by the processing state. The output routines were also very simple just like the input routines, and provided an output service. These included items such motor events, or radio broadcasts.
3.3.3 Wireless Ad-Hoc Vent Mesh Network

A lightweight simple ad-hoc vent mesh network was designed for allowing wireless communication between nodes. An ad-hoc network is one in which there is no pre-existing infrastructure or centralized control, where the network is dynamically formed and managed by the participating nodes. This is important for the SAVE network as the system control is intentionally decentralized, where the creation and topology of the network can quickly change depending upon a consumer’s preferences. The wireless network implementation can be divided up into four distinct parts:

1. The firmware driver and interface to the MRF49XA radio transceiver.

2. The radio packet protocol and construction.

3. The higher level mesh network control mechanics.

4. The vent network pairing and creation.

MRF49XA Firmware Driver

The MRF49XA radio driver was the critical path within the SAVE firmware implementation. This was caused in a large part due to the shortcomings of the transceiver’s limited receive FIFO depth of only two bytes. The MRF49XA pulls an interrupt line low when it detects an incoming transmission, notifying the MCU for service. Ideally, with a larger FIFO depth the MCU would have more time to determine when to best pull from the FIFO queue. However, the limited FIFO depth of only two bytes enforced a very rapid response time for the MCU to handle the transmission. If the MCU did not meet the service window, the radio transceiver would overwrite the FIFO buffer and set an error bit when the MCU finally would get around to emptying
the FIFO. Therefore, the radio receive event is the only part of the firmware design that exhibits preemption from the scheduler point of view.

In order to avoid overrunning the FIFO buffer, the radio receive event required immediate attention from the MCU regardless of execution state. To accomplish this, the firmware had two possible avenues for implementing the radio receive handling. The first option was to reduce execution times for the event task routines, such that when a radio receive event occurred the scheduler could switch to handling the event quicker. The longest task routines in the system were those that required communication with peripheral modules using I2C, SPI, or USART. Even at the fastest clock rates these routines still required transmitting multiple bytes and would still miss the service window for avoiding the buffer overrun. Therefore, the only way to shorten the task routine times would be to rework the I2C, SPI, USART ASF implementations to use an asynchronous method via interrupts. However, this would have been reaching in and altering the ASF framework so it was decided to not be used for the SAVE project implementation.

The SAVE project ultimately decided to go with the second option, which was to directly pull the data from the receive FIFO within the MCU ISR. Within the ISR handler the data would be emptied from the FIFO and into a buffer associated with an event task, where it would then be scheduled for processing. This introduced a preemptive approach from the scheduler point of view that broke consistency with the rest of the firmware implementation. With the exception of a radio receive event, all of the SAVE system events work under soft real-time constraints. A “soft real-time constraint” refers to a timing constraint in which a failure does not occur if the timing constraint is not met. However, a “hard real-time constraint”, such as what the radio receive event operates under, will cause a failure if the deadline is not met. Because
of this fact, the scheduler could take time away from other task routines without fear of failure as any routine that could be possibly executing is operating under soft real-time constraints. Furthermore, due to the low event rate for the SAVE system, the possibility of starvation occurring from only servicing radio receive events was not a factor.

Radio Protocol and Packet Construction

The radio protocol and packet construction revolved around the MRF49XA transmission specifications while also striving to minimizing the radio packet size. The two largest sources of power consumption after the servo motor, are radio transmit and radio receive. Thus, it was a key motivator to minimize radio transmission as much as possible in order to conserve power. A shorter packet protocol also decreases the probability of radio packet collisions, and therefore also reduces the likelihood of needing to re-transmit dropped packets.

![Figure 3.8: 14 Byte Radio Packet Structure](image)

The MRF49XA required at the start of each broadcast a series of bytes to align the clock recovery circuit on the receiver known as a preamble. The preamble is a series of 1s and 0s that the radio transceiver’s clock circuitry uses for extracting timing data from, so as to align with the incoming data transmission. The radio packet preamble was not pushed into the FIFO for the MCU to pull, and was only used by the radio for synchronization purposes. Following the preamble were two synchronization
bytes that acted effectively as a networking address. The synchronization bytes only “synchronize” with matching data, and therefore ignore any messages that do not match. For the SAVE project and prototype environment these were set to use the default values for testing. However, in a production environment these two bytes would help serve as the global network address for a residence.

After the MRF49XA had synchronized with a radio packet transmission, the start of the radio payload was delivered. For the SAVE project, the first three bytes of the radio payload consisted of the header. The header contained internal networking information such as destination address, device type, the internal sub-network ID, and the radio packet command. Following the header was a four byte field consisting of the command payload that would carry the necessary data for the associated command. The dword sized command payload was flexible enough to accommodate the needs of the SAVE system without being too large. Additionally, the firmware implementation worked to condense information if needed to fit within the command payload to keep the radio packet size minimal.

Following the command payload was a simple two byte checksum for error checking upon receipt of a transmission. In a production environment this would need to be changed to a more robust Cyclic Redundancy Check (CRC), but was sufficient for the project. Finally, the MRF49XA required a one byte pad or “dummy” byte to be sent to keep the receiving clock synchronized with the transmission while it finished pushing data into the FIFO. Similar to the radio packet preamble, the pad byte would not be seen by the MCU when it drained the FIFO and was purely for synchronization purposes.
Mesh Network Control Mechanics

An ad-hoc mesh network topology for the SAVE project was used to enable the overall system control and device interaction. Since the SAVE system is intended to compliment and not replace existing HVAC systems, the control is intentionally decentralized. For this purpose, a mesh network allows for a greater amount of flexibility to build, extend, or modify a given vent network configuration. The SAVE system network topology can be best described as a composition of several subnets, where a subnet represents a controlled temperature zone. The SAVE system allows for at most 254 controlled zones on a global network, with a maximum of 254 automated vent registers per zone.

The subnet control was provided by the zone controller devices. Each zone controller acted as both an entry point for its representative subnet, as well as contributing to the overall SAVE system control. A subnet was comprised of a single-zone controller paired with one or more automated vent registers. A zone controller therefore coordinated the state of the subnet by commanding the paired vent registers to open or close. It would additionally derive it’s subnet state through the automated vent registers detection of the HVAC operating state. This subnet state information in conjunction with the overall system state allowed for intelligent actions to be taken by the zone controller that benefited the system as a whole.

Each zone controller contributed to the management of the overall SAVE system by maintaining a table that tracked the system state and overall network health. The table provided tracking information for each subnet (or zone) present on the global network. The table data for the subnet states included the number of automated vent registers within each zone, the current HVAC state for each zone, and whether
any of the automated vent registers were closed. This snapshot of the overall network and its subnet states allowed for decentralized control, where no one zone controller dictated the logic of the system. Instead, each zone controller leveraged the overall system state information to take the best possible action given the current network conditions.

Device communication was entirely event driven by either periodic timed events or changes in the HVAC operating state. This event driven approach encouraged a low power profile as the overall system transmission and communication rates were low. For the SAVE prototype implementation, a ten second interval was used for
providing device state updates. The interval rate in a production environment would ideally be lower, anywhere from one to five minute intervals to further reduce the power budget. A faster rate was used on the prototype system to better align with the SAVE system testing chambers. These state updates effectively provide a system “pulse” that allows the zone controllers to monitor the state of the SAVE system, and overall network health. If a device stops providing updates for a given period of time, the zone controllers could then rule a device inactive and update their internal state logic accordingly.

To minimize the power consumption of the automated vent registers, data transmissions would only occur when an HVAC state change was detected or a periodic update timer had triggered. When one of these events took place, it would initiate an interaction with the paired zone controller. During the communication exchange the automated vent register would pass its updated state information to the zone controller. This state information contained the details on whether the HVAC system was heating or cooling, active or inactive, and whether the automated vent register was opened or closed. The zone controller then used this information to update its own internal state (and therefore, that of the zone), and then would respond to the automated vent register specifying the new state.

This event driven “check-in” process, shown in Figure 3.10, used by the automated vent registers allowed for the communication to only occur when necessary. It eliminated wasted transmissions and facilitated a simple transaction based state update mechanism for the zone controller and overall subnet. The SAVE prototype system did not enable the low power sleep states for the radio transceiver as system debug and testing were of higher priority. However, enabling the low power sleep states would be a simple change to implement.
The zone controllers used a simple broadcast based communication method for the mesh network interaction. The SAVE network had to ensure that two zones installed on opposite sides of a house could still communicate. In order to achieve this, the radio needed to transmit at a higher power, making radio transmissions more costly. Therefore, to limit the number of hops and retries, a broadcast based approach focused on limiting the need to re-transmit data. This allowed for a single transmission from a zone controller to be received by all zone controllers, as opposed to a series of intermediate hops (re-transmissions) to route the same information.

The subnet communication used a broadcast based approach as well, but only in the direction of the zone controller to the automated vent registers. This allowed the zone controller to communicate and interact with the automated vent registers as a group, instead of individuals. Individual commands when needed were embedded into the radio packet payload, and then parsed by the individuals to determine if any action was necessary on their part. The automated vent registers to zone controller communication was point to point.
Device Pairing and Network Creation

A deliverable for the SAVE project was to implement an automated pairing process that simplified network creation and management. The goal was to alleviate the issues associated with the Activent’s micro-switch pairing process that was seen as cumbersome and inflexible. To accomplish this, a simple button was provided on the devices to act as a pairing mechanism for the SAVE system. Similar to the Wi-Fi Protected Setup (WPS) used by Wi-Fi networks, the implementation simplified the pairing process effectively down to a user pushing a button on the new device and existing device to initiate and complete a pairing operation.

For the SAVE prototype system the pairing process and network creation was targeted for a fixed residential network. The project used the default MRF49XA address for the global network ID, but did not focus on the particulars of implementing global networks. In a production environment the system would need to be able to have a unique global network for each residence. However, testing for the SAVE prototype system was performed in a mock chamber of a single residence, so was unnecessary for the project implementation.

A simple algorithm was established for defining an unsecured device pairing process and network creation. The pairing process was initiated by first pressing the “pair” button on the existing device or desired network to join. Internally, the device would switch to a predefined “join” network apart from the current global network. The device then would randomly generate a unique join ID to use for identification during the pairing process. Once the device had successfully configured to communicate on the join network, it would then send out a join initialize radio broadcast. The device then would enter a timed state to wait for another join initialize
broadcast issued by the device to be paired with. The timeout for receiving a join initialize broadcast was set to 10 seconds, after which the pairing process would fail and the device would restore its original settings to the global network.

After starting the pairing process by pushing the pair button on the first device, the same sequence would be applied to the second device. The pair button would be depressed, the device would switch to the join network while creating a unique join ID, and then the device would issue a join initialize radio broadcast. After issuing the join initialize broadcast, the device would similarly enter a timed state to wait for the next join initialize broadcast event. This time however, the join initialize broadcast sent from the second device is received by the first device that had been in the wait state. The first device, upon receipt of the second devices broadcast of join initialize, would then take the necessary steps to begin completing the pairing process.

There are three different kinds of pairing operations that dictate the pairing completion procedures taken by the devices. The three types of pairing operations that can occur during a pairing event happen between a zone controller and an automated vent register, between two automated vent registers, or between two zone controllers. Depending upon the pairing interaction, different steps need to be taken to complete the pairing process. Any failed pairing attempt would resolve in the restoration of the original networking state of the device.

**Pairing Between Zone Controllers and Automated Vent Registers**

The most simple pairing completion process occurs between a zone controller and an automated vent register. In this pairing scenario, the automated vent register will always join the zone controller’s subnet network. This is because an automated vent register always belongs to a subnet, and can only be paired with a single-zone
controller. If a zone controller were allowed to join the automated vent registers subnet, there would be two zone controllers on the same subnet which would violate the mesh network topology. Therefore, the vent registers always join the zone controller’s subnet.

Depending on which device’s pair button was pushed first, the network configuration exchange and pairing completion process differs slightly. The first scenario is the automated vent register’s pairing button was pressed first, in which case it receives the zone controller’s broadcast of join initialize. In this case, the automated vent register acquires the zone controller’s network information via the payload in the join initialize broadcast. After updating its configuration to join the zone controller’s

Figure 3.11: Device Pairing Sequence Diagram
network, the automated vent register would send a join sync command that would complete the pairing process.

The second scenario is that the zone controller’s pairing button was pushed first, in which case the zone controller receives the automated vent registers join initialize broadcast. Here, the zone controller would simply populate the join sync command with the necessary network configuration information needed by the automated vent register. The automated vent register would then receive the sync command, update its configuration accordingly, and the pairing process would be completed.

**Pairing Between Two Automated Vent Registers**

The pairing completion process that occurs between two automated vent registers is similar to the process between a zone controller and an automated vent register. The automated vent register whose button was first pushed provides the target subnet to join for the second device. When the first automated vent register receives the join initialize broadcast from the second automated vent register, it responds with the current networking information in a join sync command.

When the join sync command is received by the second vent register, it then prepares to effectively ask permission to join the subnet by contacting the subnet zone controller. This request to the zone controller is a join configure vent broadcast. The zone controller, which maintains a table of automated vent register IDs for the subnet, responds to the configure vent broadcast with a new unique subnet ID for the second vent register to complete the pairing process. If the subnet however is full and cannot accept any more automated vent registers, the zone controller declines the request for the second vent register to join, and therefore cancels the pairing process.
Pairing Between Two Zone Controllers

The most rigorous pairing completion process occurs when two zone controllers are paired. This is because when two zone controllers are paired, their respective subnets including all of their paired vent registers must migrate over and onto the new network. The first zone controller that initiated the pairing process receives the join initialize broadcast sent from the second zone controller. The first zone controller then responds with its network information via a join sync command.

When the second zone controller receives the join sync command containing the network information for the first zone controller, it begins to attempt joining the new global network. The second zone controller attempts to join the new global network by essentially looking for an address vacancy that it can use to register with. The zone controller issues a configure zone broadcast for each address to check if the address is occupied. If the zone address is currently being used by another zone controller, that zone controller will respond with an address in use response. If the address is vacant however, the zone controller will continue to issue a series of short checks to ensure that the configure zone broadcast was not missed. If the address is still vacant following the checks, the zone controller will then register with the vacant address. After registering the new global network address, the pairing process is complete. If however the zone controller was unable to find an address vacancy on the new global network the pairing process between the two zone controllers fails.

When a successful pairing process has occurred between two zone controllers, the second zone controller must ensure to migrate its vent registers from the previous network onto the new network. To accomplish this, the zone controller maintains a small presence in the old network to respond to any communications issued by its
paired vents. This allows for the zone controller to send address update commands to the paired vent registers to move onto the new network. Once all vents have successfully move onto the new network, the zone controller then drops the multi-network presence and operates solely on the new network. The zone controller only maintains a presence in the old network for some max time duration (two minutes for the prototype implementation), whereupon any vent register that has not checked in before then will lose their pairing role with the zone controller.

### 3.3.4 HVAC System State Detection

The HVAC system state detection was accomplished by having the automated vent registers monitor the rate of change for the pressure readings in conjunction with temperature data. Initially the desire was to have the Freescale MPL3115A2 collect the pressure and temperature samples autonomously while keeping the MCU in a low power sleep state. The MPL3115A2 has programmable thresholds that can trigger interrupts when hit. It also has a very deep sample FIFO that periodically takes sensor measurements at programmable intervals. However, because the MPL3115A2 pressure sensitivity was not robust enough to reliably detect the pressure variations, this functionality could not be leveraged. Therefore, it was left to the MCU and firmware implementation to incorporate a more advanced analysis of the sample data in order to detect the HVAC state changes.

The role of the pressure sensor was to detect the pressure changes within the air duct system when the HVAC turned on or off. Taking the approach of directly measuring the pressure changes was not feasible as any change fell within the margin of system noise. The SAVE prototype system worked around this limitation by taking a more proactive approach while applying a simplistic algorithm to the sampling data.
Since the MPL3115A2 did not have the sensitivity to directly detect the pressure changes, the rate of change for the pressure samples was measured instead by the firmware. This meant that instead of looking for a sloping trend indicating an HVAC system state transition from on to off, the firmware was instead looking for a spike of rapid change. To help facilitate this rapid change, the automated vent registers could effectively test the HVAC system state by merely opening or closing the vent louvers.

By opening or closing the vent louvers a very large pressure spike was created at the interface that could easily be detected by the MCU using the pressure sampling algorithm. The pressure sampling algorithm was simply a rolling average of the pressure samples rate of change. For the project, the MPL3115A2 sensor was configured to take pressure and temperature readings every second. The pressure sampling algorithm maintained a five sample buffer (a five second window) for applying the rolling average to. A threshold was empirically chosen for the SAVE prototype system implementation that would reliably identify HVAC activity state changes. It is important to note that any false positives triggered by the threshold for an HVAC state change are not detrimental to the operation of the overall SAVE system. In fact, most false positives for an HVAC activity state change will eventually self correct over time. Those that do not self correct are just included in the current control logic, meaning they are being actively managed but do not hurt the system.

The HVAC mode of heating vs cooling was handled by tracking the temperature data over a five minute period after a pressure event had occurred. Directly after the pressure event, a snapshot of the temperature data was taken and then would be later compared against the temperature taken after the five minute interval. If the temperature increased between the two samples the HVAC system was in a heating mode, otherwise a temperature decrease indicated the HVAC system was cooling. For
the majority of cases this is a reliable detection method, where the ambient air duct temperature is going to be different than the conditioned air.

During winter time the ambient air duct temperature will be much colder than the conditioned air, and vice versa in the summer time. However, during transitional seasons the HVAC heating versus cooling mode will be more difficult to reliably detect. This is because ambient air temperature could be near the conditioned air temperature, disrupting the baseline from which to measure from. A more advanced system level approach would need to be implemented in a production environment to handle this case. For the SAVE prototype implementation and testing chambers however it was an adequate solution.

3.3.5 HVAC Temperature Balancing and System Safety

The temperature balancing algorithm dictated the high level control for the SAVE system network. As the SAVE system was designed to be decentralized, the algorithm was implemented from the perspective of each zone controller taking the best possible action given a current overall system state. As each zone controller contained the tabled information containing the key characteristics of each subnet, it could calculate the best possible action for its own subnet autonomously. This meant that while each zone was acting in a selfish behavior, the behavior was consistent with and beneficial to the system as a whole.

A chief concern for the SAVE project was the potential of damaging the HVAC unit, either by icing the coils or triggering the high-limit switch. Coil icing on HVAC units can in some cases cause serious damage, as is the same for when a furnace begins to operate on the high-limit switch. Both of these instances are caused by closing too many air vent registers when redirecting air flow to other zones or areas
within a household. A goal for the SAVE prototype implementation was to reduce the probability of this damage occurring while it was opening and closing the air vent registers. If the SAVE system closed too many vent registers it could begin to adversely affect the system and cause the aforementioned damage.

The temperature balancing algorithm worked to fulfill both evenly distributing the temperatures across the active zones while also maintaining the register close limitation for the HVAC system safety. The first step of the algorithm was identifying the currently active zones on the SAVE system. An active zone was considered to be one in which the HVAC system was on and delivering conditioned air. The zones that were active would participate in the SAVE system temperature balancing logic. For zones that were identified as inactive, they would set all of their vent registers open except for one to detect the HVAC system on pressure change.

For the prototype system implementation and testing chambers, detecting an HVAC system change by keeping an automated vent register closed was sufficient. However, in a production environment a more robust implementation would be needed. In some cases an HVAC system could be slow to ramp on, and may not be quick enough to trigger the pressure threshold for detecting the HVAC system change. In this case, a rotating check would need to be implemented in each zone to periodically verify that the zone will still active. The rotation check would simply change the closed vent register between the various vents within the same subnet. This in turn would ensure verification of the HVAC system state. Unfortunately, this rotating verification method consumes more energy but would be necessary when using a low cost pressure sensor similar to that of the MPL3115A2.

Once a zone controller has identified as being active, it participates in the overall SAVE system temperature balancing algorithm. Each zone controller first consults
its network table to retrieve the current state of the SAVE network and the various subnets. The zone controller then sorts all of the active zones according to their last reported temperatures.

![Temperature Balancing Example](image)

Figure 3.12: Temperature Balancing Example

After all of the active zones had been sorted by their last reported temperatures, the zone controller then counts the total number of automated vent registers that are managed by the active zones. This allows for the zone controller to identify where the register close limit should be given the current number of vent registers in active zones. For the SAVE prototype system, a register close limit was chosen to be 50% of the active zones. Once the close limit had been applied to the active vent register count, the zone controller could then determine the open and close count for each zone on the system. The zone controller would then simply update its own subnet to match the derived open and close counts for the automated vent registers found during processing.

This algorithm essentially directs the conditioned air into the rooms with the greatest temperature deltas. For example, Figure 3.12 depicts a heating scenario where the coldest rooms will be targeted to receive heated air first. The register vent
close limitation dictates the overall vent to open/close ratio for the active zones. In this case, there are three zones that are actively participating in the temperature balancing algorithm. There are a total of 10 vent registers within these active zones. Using the register close limit of 50%, each zone controller concludes that there should be 5 vents open to 5 vents closed. Since the zones have already been sorted by temperature, the zone with the greatest temperature delta receives priority in opening its vents first (in this case, Zone A). Then the next lowest temperature zone get the next selection of opening its vent registers and so on until the open vent register limit is hit. The remaining zones then close their automated vent registers as calculated by the algorithm.
CHAPTER 4

SYSTEM TESTING

4.1 Test Setup

There were two testing chambers that were built to test the SAVE system implementation. The first testing chamber, shown in Figure 4.1, was designed for developing and testing the temperature balancing algorithm. The testing chamber was composed of four separate boxes made of Plexiglas representing the different rooms or zones within a residence. Each of these four boxes were sized differently so that their rates of heating and cooling would simulate the varying thermal characteristic between zones seen in homes. The HVAC system heating was simulated by fitting a hair dryer to a PVC air duct system that was routed to each of the individual boxes.

The second testing chamber, shown in Figure 4.2, utilized the same Plexiglas boxes for zone simulation but replaced the PVC ductwork and hair dryer with an actual HVAC unit and metal air duct. This chamber was designed for evaluating the HVAC system state detection for the automated air vent registers, as well as the energy efficiency and system health effects associated with opening and closing the registers.
Figure 4.1: Temperature Balancing Testing Chamber

Figure 4.2: HVAC System State Testing Chamber
4.2 HVAC Temperature Balancing

The temperature balancing testing chamber only looked at heating scenarios and was not used for testing cooling conditions. The SAVE system was also configured to have all zones active by default, and set the temperature balance algorithm for an HVAC heating mode. Measurements were first made by completely disabling the SAVE system and taking temperature measurements with all of the air vents open as shown in Figure 4.3. This represented the typical configuration for a residence, where all air vent registers are left open by default. Over a one minute interval during an applied heating cycle the temperature delta between the zones drifted to as much as 12°C, with a minimum temperature delta of 3°C.

![Temperature Variation with SAVE System Disabled](image)

Figure 4.3: Temperature Variation with SAVE System Disabled

The SAVE system was then enabled and measurements again taken while a heating cycle was applied to the zones as shown in Figure 4.4. With the SAVE system enabled, the temperature deltas between the zones at any given point in time was held within 1°C. This was a dramatic improvement and validated the SAVE system temperature balancing implementation.
4.3 HVAC System State Detection

The testing chamber for the HVAC system state detection incorporated all aspects of the SAVE system control. It was primarily used for verifying the HVAC state detection by the automated vent registers, but also tested the overall temperature balancing mechanisms as well. The testing chamber looked at both the heating and cooling scenarios without a pre-configured SAVE system orientation. The SAVE prototype implementation was able to both successfully detect an HVAC on/off state transition, as well as determine whether the HVAC unit was in a heating or cooling mode. The temperature balancing algorithm was also again validated as the temperature deltas between the zones were held within 1°C.
CHAPTER 5

CONCLUSIONS

The project was able to accomplish all four of the specified project deliverables for the SAVE prototype system. First, an electrical board was successfully created that enabled wireless system communication and the facilities for controlling the automated vent louvers. This provided the platform under which the SAVE system firmware implementation could run against. All of the project hardware design goals were also achieved for the hardware design. The board was very easy to debug and characterize using with the exposed debug headers and power jumpers available on the board. The dimension were under 3.5” x 2.5”, and an embedded PCB antenna with over 150 ft NLOS were successfully implemented.

The second project deliverable was also successfully accomplished and delivered the necessary control firmware for the SAVE project. This included an ad-hoc mesh network that utilized a low interval communication method for coordinating the actions of the SAVE system. The goal of implementing a compressed radio packet protocol for device communication was mostly met. An improvement could have been made by utilizing the hardware CRC module provided by the ATxmega64D4 instead of a simple checksum for better packet integrity. Additionally, in a production environment a more robust radio stack would need to be developed to optimize the mesh network power. Overall however, the project was guided by low power design
considerations and implemented when possible.

The SAVE system control also took precautions to limit any damage occurring to the HVAC system by monitoring the number of closed vent registers during operation. Ensuring that the SAVE system did not damage the existing system is a very important characteristic of the system, and was an important goal that was accomplished. The goal for creating a portable architecture to minimize code rework when switching platforms was mostly accomplished. However, because the ASF was treated as the bottom most layer below the PAL, the PAL ended up being somewhat tied to the ASF platform (or Atmel MCUs). In hindsight, there should have been a translation layer between the PAL and ASF to help keep them decoupled.

The third project deliverable for creating an automated pairing process was also achieved. The pairing process was implemented as a simple sequence of pushing buttons on the desired devices to pair, whereupon the underlying pairing details would be automatically handled for the user. Additionally, the pairing process would also resolve moving any vent registers paired to a zone controller onto the new network if necessary. This met another design goal for the SAVE project, saving the user from having to individually re-pair all the vent registers when moving a zone controller onto a new network.

Finally, the fourth and last project deliverable of detecting the HVAC system state was also successful. A simple running algorithm was applied to the derivative of the pressure sensor data, where a spike indicated an HVAC system on/off state change. This spike would break a threshold barrier notifying the automated vent register that an HVAC system state change had occurred. While the threshold value was ideal for the testing chambers used on the prototype system, most likely this would not be a reliable method in a production environment. However, the design
of the SAVE system allowed for false positives triggered by HVAC state changes to either self correct over time or be safety included in the SAVE system control logic.

The project goal of using firmware to compensate for low cost components was mostly successful. The largest impact of choosing a cost reduced implementation came with a trade off in power efficiency. The project was able to demonstrate that it was feasible for firmware to overcome the hardware limitations, but this usually came at the cost using more power. Generally speaking, the absence of deeper FIFOs and more functionality shifted the responsibility over to the MCU to resolve. In addition, the limited sensor resolution necessitates the need for the automated vent registers to periodically check the HVAC system state by modulating their vent louvers. Again, this results in a waste of power efficiency that could have been saved had the pressure sensor supported a higher sensitivity.

Overall, the project proved the viability of the SAVE system as an affordable solution to the centralized thermostat issue. The SAVE system is able to highly regulate temperature variations throughout multiple zones without endangering the HVAC system safety. In addition, the project was also able to demonstrate the ease in which the SAVE system could be configured and built. A solution such as the SAVE system could set the new standard for residential comfort, while providing easy installation and configuration at a low cost.
REFERENCES


APPENDIX A

PROTOTYPE BOARD PO AND BOM

Figure A.1: The Gold Phoenix PCB PO
<table>
<thead>
<tr>
<th>#</th>
<th>Designator</th>
<th>Quantity</th>
<th>Value</th>
<th>Manufacturer</th>
<th>MPN</th>
<th>Supplier</th>
<th>Supplier Part No.</th>
<th>Description</th>
<th>Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>R1, R2</td>
<td>2</td>
<td>330Ω</td>
<td>Stackpole Electronics</td>
<td>RMC0402FT330R</td>
<td>Digi-Key</td>
<td>RMC0402FT330RCT-ND</td>
<td>Thick film resistor, 1/16W, ±1%</td>
<td>0402</td>
</tr>
<tr>
<td>2</td>
<td>R3, R4, R5, R6, R7, R8, R9</td>
<td>7</td>
<td>220Ω</td>
<td>Rohm Semiconductor</td>
<td>MC01018FT2200</td>
<td>Digi-Key</td>
<td>RHM220DCDT-ND</td>
<td>Thick film resistor, 1/16W, ±1%</td>
<td>0402</td>
</tr>
<tr>
<td>3</td>
<td>R10, R11</td>
<td>2</td>
<td>1Ω</td>
<td>Stackpole Electronics</td>
<td>RNC0603FT100R</td>
<td>Digi-Key</td>
<td>RNC0603FT100DRK-ND</td>
<td>Thin film resistor, 1/8W, ±1%</td>
<td>0803</td>
</tr>
<tr>
<td>4</td>
<td>R12, R13</td>
<td>2</td>
<td>1kΩ</td>
<td>Rohm Semiconductor</td>
<td>RMC0402FT1K0</td>
<td>Digi-Key</td>
<td>RMC0402FT1000CT-ND</td>
<td>Thick film resistor, 1/16W, ±1%</td>
<td>0402</td>
</tr>
<tr>
<td>5</td>
<td>R14</td>
<td>1</td>
<td>100kΩ</td>
<td>Stackpole Electronics</td>
<td>RMC0402FT100K</td>
<td>Digi-Key</td>
<td>RMC0402FT100KCT-ND</td>
<td>Thick film resistor, 1/16W, ±1%</td>
<td>0402</td>
</tr>
<tr>
<td>6</td>
<td>C1, C2, C3, C4, C5, C6, C7, C8, C9, C10</td>
<td>10</td>
<td>10nF</td>
<td>Kemet</td>
<td>CD0402NCT4</td>
<td>Digi-Key</td>
<td>CD0402K474RACTU</td>
<td>Ceramic capacitor, X7R, 18V, ±10%</td>
<td>0402</td>
</tr>
<tr>
<td>7</td>
<td>C11, C12</td>
<td>2</td>
<td>10μF /10V</td>
<td>TDK Corporation</td>
<td>C2116X7R106M160AC</td>
<td>Digi-Key</td>
<td>C2116X7R106M160AC</td>
<td>Ceramic capacitor, X7R, 10V, ±20%</td>
<td>1206</td>
</tr>
<tr>
<td>8</td>
<td>C13, C14, C15</td>
<td>3</td>
<td>10μF /6.3V</td>
<td>TDK Corporation</td>
<td>C212X630J106M125AB</td>
<td>Digi-Key</td>
<td>C212X630J106M125AB</td>
<td>Ceramic capacitor, X6S, 63V, ±10%</td>
<td>0803</td>
</tr>
<tr>
<td>9</td>
<td>C16</td>
<td>1</td>
<td>100μF</td>
<td>Murata</td>
<td>GRM18R871H102KA010</td>
<td>Digi-Key</td>
<td>GRM18R871H102KA010</td>
<td>Ceramic capacitor, X7R, ±15%</td>
<td>0803</td>
</tr>
<tr>
<td>10</td>
<td>C17</td>
<td>1</td>
<td>10000pF</td>
<td>Kemet</td>
<td>CD0402NCT4</td>
<td>Digi-Key</td>
<td>CD0402K474RACTU</td>
<td>Ceramic capacitor, X7R, 50V, ±10%</td>
<td>0803</td>
</tr>
<tr>
<td>11</td>
<td>C18</td>
<td>1</td>
<td>33pF</td>
<td>Samsung</td>
<td>C10C330JB89NNNC</td>
<td>Digi-Key</td>
<td>C10C330JB89NNNC</td>
<td>Ceramic capacitor, X7R, ±15%</td>
<td>0803</td>
</tr>
<tr>
<td>12</td>
<td>C19</td>
<td>1</td>
<td>2.2μF</td>
<td>TDK Corporation</td>
<td>C1608X631A225X080AR</td>
<td>Digi-Key</td>
<td>C1608X631A225X080AR</td>
<td>Ceramic capacitor, X7R, ±10%</td>
<td>0803</td>
</tr>
<tr>
<td>13</td>
<td>C20, C21</td>
<td>2</td>
<td>2μF</td>
<td>Murata</td>
<td>GRM1555S102RCA010</td>
<td>Digi-Key</td>
<td>GRM1555S102RCA010</td>
<td>Ceramic capacitor, X7R, ±10%</td>
<td>0803</td>
</tr>
<tr>
<td>14</td>
<td>L1</td>
<td>1</td>
<td>220Ω ±10 MHz</td>
<td>Murata</td>
<td>BU2156882215SN</td>
<td>Digi-Key</td>
<td>BU2156882215SN</td>
<td>Ferrite chip, 220Ω ±100MHz, 200mA</td>
<td>0402</td>
</tr>
<tr>
<td>15</td>
<td>XC1</td>
<td>1</td>
<td>9.1K-10.0000MEEJ-T</td>
<td>TDK CORPORATION</td>
<td>SC10-0000MEEJ-T</td>
<td>Digi-Key</td>
<td>SC10-0000MEEJ-T</td>
<td>Ceramic capacitor, 100Ω ±10%</td>
<td>0402</td>
</tr>
<tr>
<td>16</td>
<td>U1</td>
<td>1</td>
<td>ATXMEGA404-AU</td>
<td>Atmel</td>
<td>ATXMEGA404-AU</td>
<td>Digi-Key</td>
<td>ATXMEGA404-AU</td>
<td>ATXMEGA404-AU</td>
<td>AVR 8-bit RISC MCU, 64kB flash</td>
</tr>
<tr>
<td>17</td>
<td>U2</td>
<td>1</td>
<td>MRF9492A-1/ST</td>
<td>Microchip</td>
<td>MRF9492A-1/ST</td>
<td>Digi-Key</td>
<td>MRF9492A-1/ST-ND</td>
<td>RF MCU 433/868/915</td>
<td>16-TSSOP</td>
</tr>
<tr>
<td>18</td>
<td>U3</td>
<td>1</td>
<td>MPL3115A2</td>
<td>Freescale</td>
<td>MPL3115A2</td>
<td>Digi-Key</td>
<td>MPL3115A2-ND</td>
<td>Altimeter MCU 50 - 110kPa, 0°C</td>
<td>0402</td>
</tr>
<tr>
<td>19</td>
<td>U4</td>
<td>1</td>
<td>NCP4177T3P03T3G</td>
<td>ON Semiconductor</td>
<td>NCP4177T3P03T3G</td>
<td>Digi-Key</td>
<td>NCP4177T3P03T3G</td>
<td>LDO voltage regulator, 3.3V, 1A</td>
<td>SOT-233</td>
</tr>
<tr>
<td>20</td>
<td>U5</td>
<td>1</td>
<td>NCP4177T5P03T3G</td>
<td>ON Semiconductor</td>
<td>NCP4177T5P03T3G</td>
<td>Digi-Key</td>
<td>NCP4177T5P03T3G</td>
<td>LDO voltage regulator, 5.0V, 1A</td>
<td>SOT-233</td>
</tr>
<tr>
<td>21</td>
<td>BTN1, BTN2</td>
<td>2</td>
<td>SKRAA1A010</td>
<td>ALPS</td>
<td>SKRAA1A010</td>
<td>Digi-Key</td>
<td>SKRAA1A010</td>
<td>SMD Tactile Switches 6.2x6.2x4.3mm 3.52N switch</td>
<td>SOT-223</td>
</tr>
<tr>
<td>22</td>
<td>J1</td>
<td>1</td>
<td>154-13820-E</td>
<td>Kobiconn</td>
<td>154-13820-E</td>
<td>Digi-Key</td>
<td>154-13820-E</td>
<td>SMD Mini USB type B connector, 5 pin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>23</td>
<td>J2, J3, J4</td>
<td>3</td>
<td>22-28-4030</td>
<td>Molex</td>
<td>22-28-4030</td>
<td>Digi-Key</td>
<td>WS00014-03-ND</td>
<td>Breakaway header, 2.54 mm, 1 row, 3 contacts, tin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>24</td>
<td>J5, J6</td>
<td>2</td>
<td>10-89-7091</td>
<td>Molex</td>
<td>10-89-7091</td>
<td>Digi-Key</td>
<td>WS00022-10-N</td>
<td>Breakaway header, 2.54 mm, 2 rows, 10 contacts, tin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>25</td>
<td>J7, J8</td>
<td>2</td>
<td>10-89-7091</td>
<td>Molex</td>
<td>10-89-7091</td>
<td>Digi-Key</td>
<td>WS00022-10-N</td>
<td>Breakaway header, 2.54 mm, 2 rows, 10 contacts, tin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>26</td>
<td>J9</td>
<td>1</td>
<td>10-89-7141</td>
<td>Molex</td>
<td>10-89-7141</td>
<td>Digi-Key</td>
<td>WS00022-10-N</td>
<td>Breakaway header, 2.54 mm, 1 row, 3 contacts, tin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>27</td>
<td>RJMP, BJMP, BAT, LUMP, PTJMP</td>
<td>5</td>
<td>22-28-4020</td>
<td>Molex</td>
<td>22-28-4020</td>
<td>Digi-Key</td>
<td>WS00014-02-ND</td>
<td>Breakaway header, 2.54 mm, 1 row, 2 contacts, tin header</td>
<td>SOT-223</td>
</tr>
<tr>
<td>28</td>
<td>PLED</td>
<td>1</td>
<td>LTST-C151KGT</td>
<td>Lite-On</td>
<td>LTST-C151KGT</td>
<td>Digi-Key</td>
<td>LTST-C151KGT</td>
<td>SMD green clear LED</td>
<td>0805</td>
</tr>
<tr>
<td>29</td>
<td>LED1, LED2</td>
<td>3</td>
<td>LTST-C151KGBT</td>
<td>Lite-On</td>
<td>LTST-C151KGBT</td>
<td>Digi-Key</td>
<td>LTST-C151KGBT</td>
<td>SMD red clear LED</td>
<td>0805</td>
</tr>
<tr>
<td>30</td>
<td>LED3, LED4, LED5</td>
<td>3</td>
<td>LTST-C151K5KT</td>
<td>Lite-On</td>
<td>LTST-C151K5KT</td>
<td>Digi-Key</td>
<td>LTST-C151K5KT</td>
<td>SMD yellow clear LED</td>
<td>0805</td>
</tr>
</tbody>
</table>