Building an Open Source IoT Garage Controller

|  |  |
| --- | --- |
| Daniel A. Zajac  Department of Electrical and Computer Engineering  University of Michigan  danzajac@umich.edu | Jackson Harmer  Department of Electrical and Computer Engineering  University of Michigan  jharmer@umich.edu |

*Abstract*— Internet connected technologies have become mainstays of the modern household. From Internet of Things (IoT) connected coffee makers to sophisticated adaptive climate control systems, inexpensive wireless technology and popularity of voice activated digital assistants has enabled a wide variety of connected tech. Most of this technology remains closed source however. Many of the more popular connected technologies, such as Nest or ecoBee, rely on closed source protocols and cloud service backends. What happens when these companies go out of business or shut down older services? VueZone shutdown it’s services leaving owners with severely crippled expensive IoT cameras. [1] This paper focuses on building a proof of concept IoT connected garage door real time controller and fully published interface and source: The GarageRTC! The features, design, and implementation of a reference architecture built on an ESP32 microcontroller and FreeRTOS software as well as performance and possible improvements.

Keywords—GarageRTC, IoT, Embedded, FreeRTOS, ESP32, NodeMCU, RTOS

# Introduction

As IoT become more common place in our daily lives we become increasingly dependent on the connected services that support those systems. Some of the more popular devices, such as Nest thermostat, are the product of small teams operating as a startup. As these teams grow they are sometimes bought out by larger corporations interested in entering into the IoT market. For some less fortunate start‑ups, they never break through into profitability and slowly descend into obscurity.

What happens to the devices that depend on the cloud services previously maintained and supported by those teams? Sometimes they can continue to function but with crippled or limited performance. Often, they are rendered useless as registration and web-based configuration tools become obsolete. For inexpensive devices, they can be cannibalized for parts or simply disposed of. For more expensive equipment, such as the Juicero, the founding company dissolved leaving many users with useless $400 IoT juice machines. [2]

The only guaranteed way to ensure these devices can be indefinitely supported is to implement an open source methodology including web APIs. Preferably, a completely open source hardware and software reference design would be created and published. Startups wishing to develop derivative works, could leverage the technology could then extend the designs. If the startup dissolves or can no longer support their design, as long as compatibility with the reference design was maintained, an internet user community would be able to continue support. This concept has been demonstrated profitable and sustainable in large software efforts such as IBM supported Red Hat Linux. [3]

To prove out this concept, the research team designed and built a reference design for an IoT connected Garage real time system, or simply, the GarageRTC.

The GarageRTC is an automation system for a consumer garage connected as an IoT device, Figure 1. The system makes the status of garage available to the user connected remotely through a web-based interface or locally via a display and control panel. Inside the garage, the system is connected to the garage door opener, a garage light, and an alarm. The system uses sensors to detect door position, accumulation of carbon monoxide, temperature, and objects in the path of the door. From the control panel or web interface, the user can check the status or manipulate the controls.

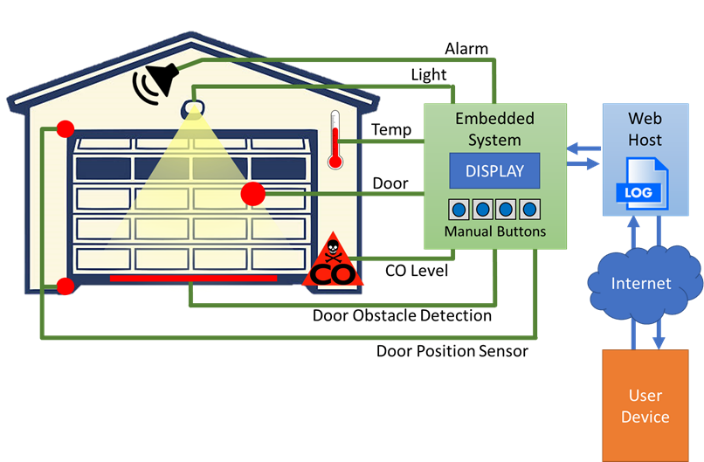


Figure 1. GarageRTC Concept Diagram

This paper documents the design process, development, implementation, and testing of a WiFi connected garage monitoring system. It implements a basic JSON based API and includes a reference web application that can easily be hosted in a personal cloud or Linux based development board.

## Organization of this Paper

This paper is organized into ten parts:

* Introduction – Introduces the motivations and organization of this project.
* Objectives and Functional Description – Outlines the high level objectives for the project and reviews the design process employed.
* Requirements – Reviews the high level requirements the GarageRTC was designed to meet.
* High Level Design – Describes the process used to decompose and partition the system and generate high level designs and test plans.
* Detail Design – Discussion on the detailed design decisions including selection of hardware, software, and web application.
* Testing – Review of the tests and stages used to ensure proper functioning of the system.
* Accomplishments – Review of the objectives met, limitations of the current design, possible improvements, and lessons learned.
* Comparison of Commercial Offerings – A comparative discussion about similar products commercially available.
* Conclusions and Closing Remarks
* References

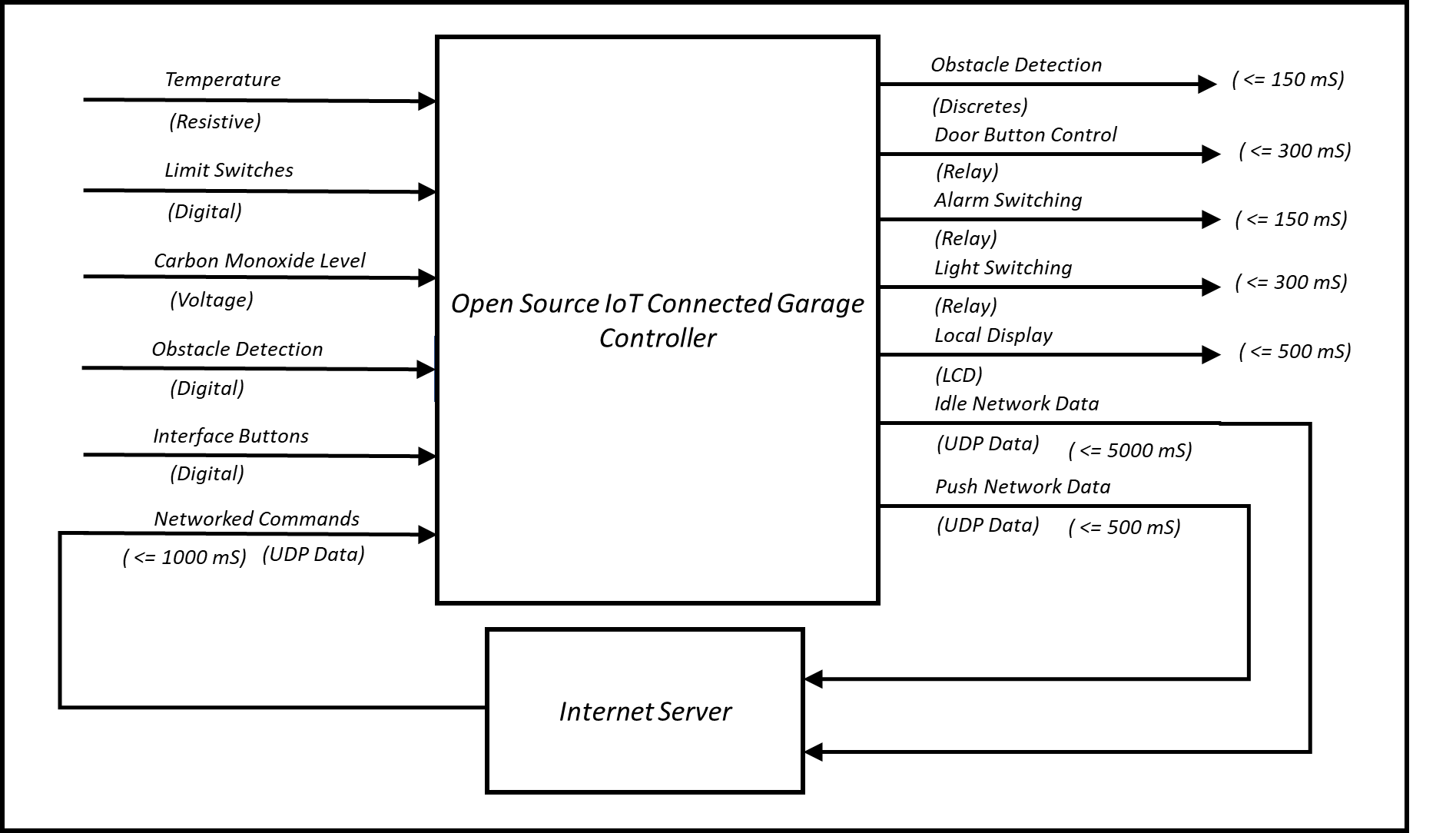
# Objective and Functional Description

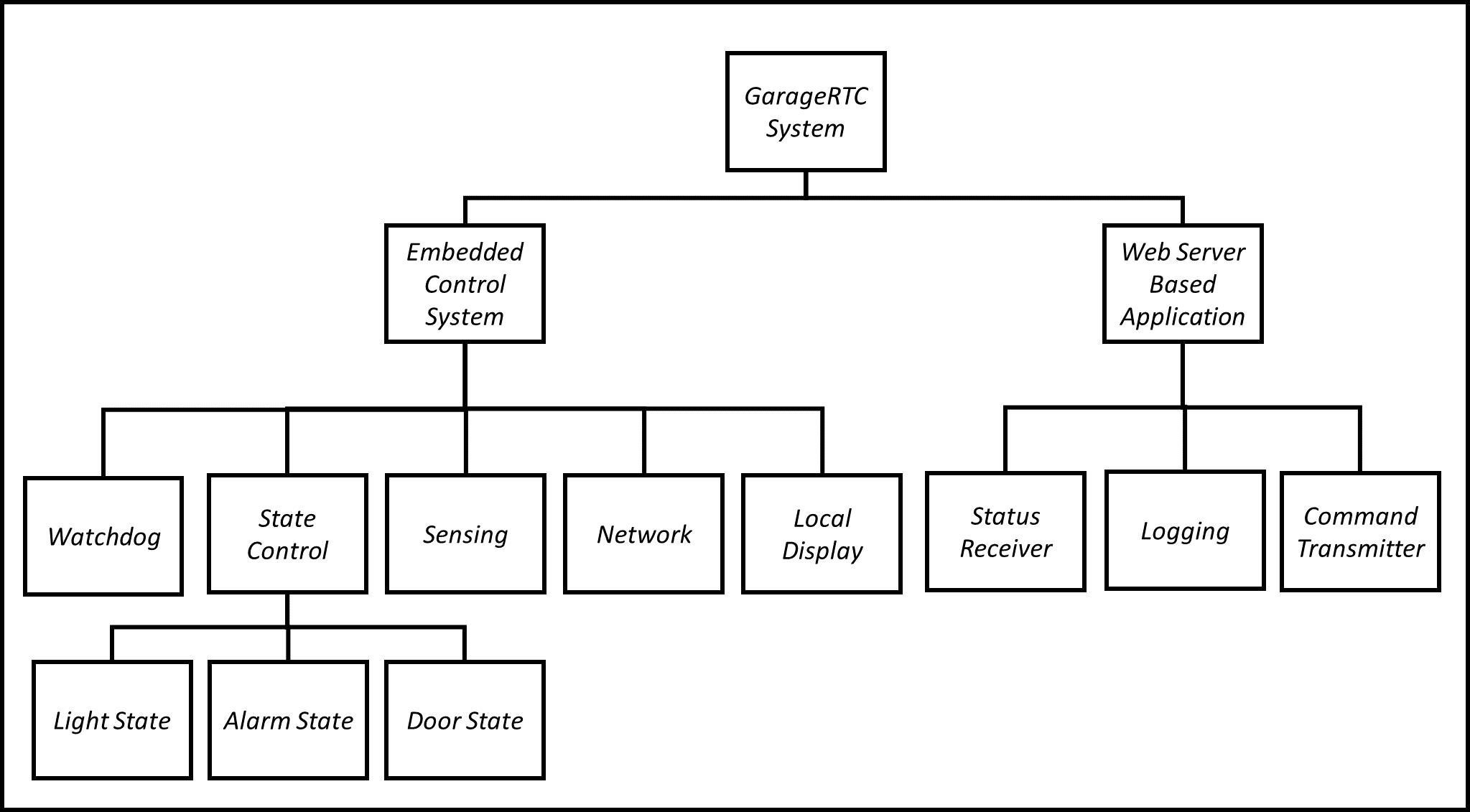
## Design Process

This project was developed using a waterfall

# Requirements

# High Level Design





## Embedded System

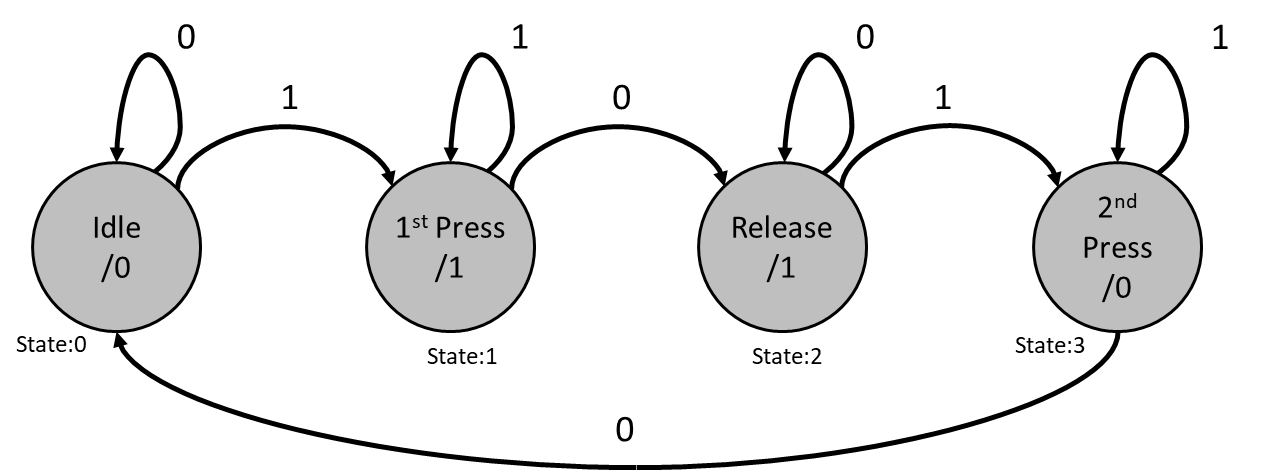
For the embedded system, a real-time OS was necessary to provide predictable response to inputs while simultaneously executing multiple tasks in a timely fashion. While scratch development of a RTOS was considered, ultimately the FreeRTOS project was selected. FreeRTOS implements a real-time kernel and schedulers targeted as low resource microcontrollers. The project is professionally developed and is available for free use in commercial embedded systems. It supports a wide variety of hardware and offers excellent documentation and, most importantly, is freely available including source. [4]

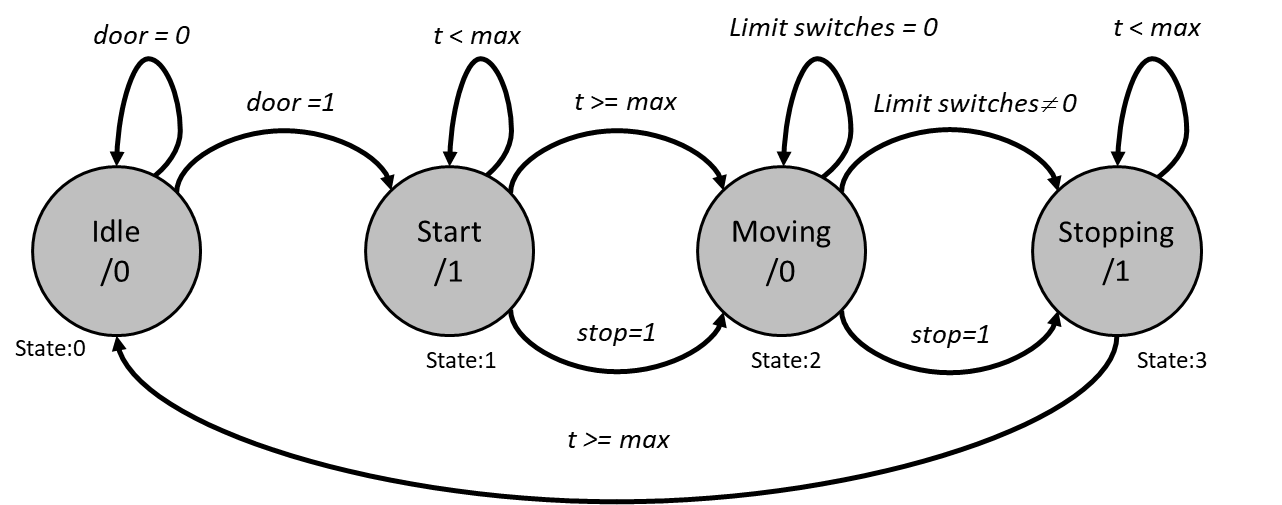
## Web Application

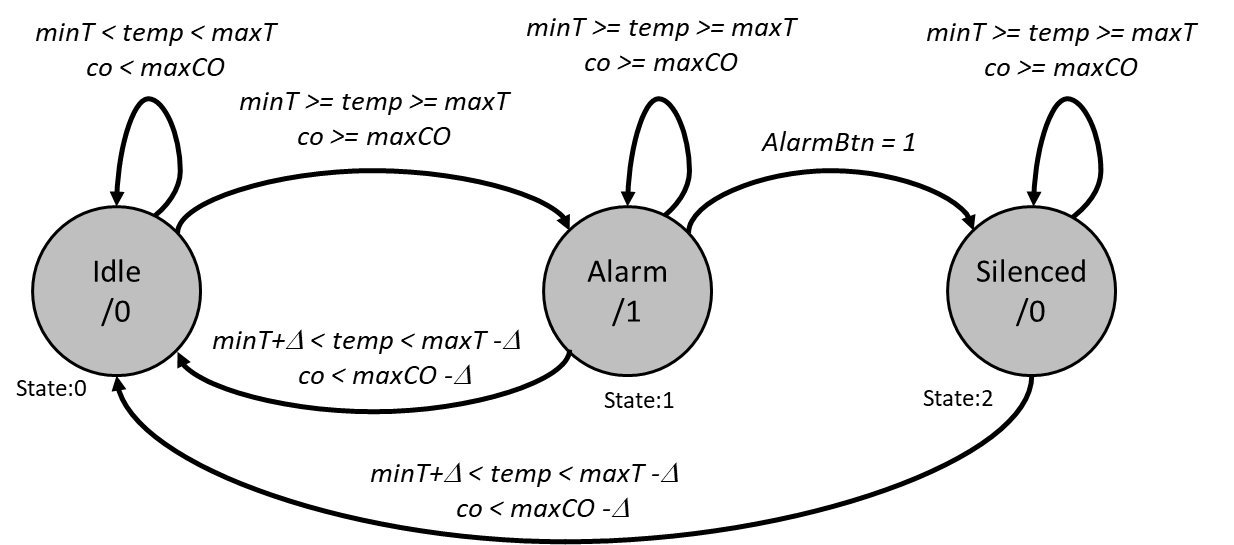
# Detail Design

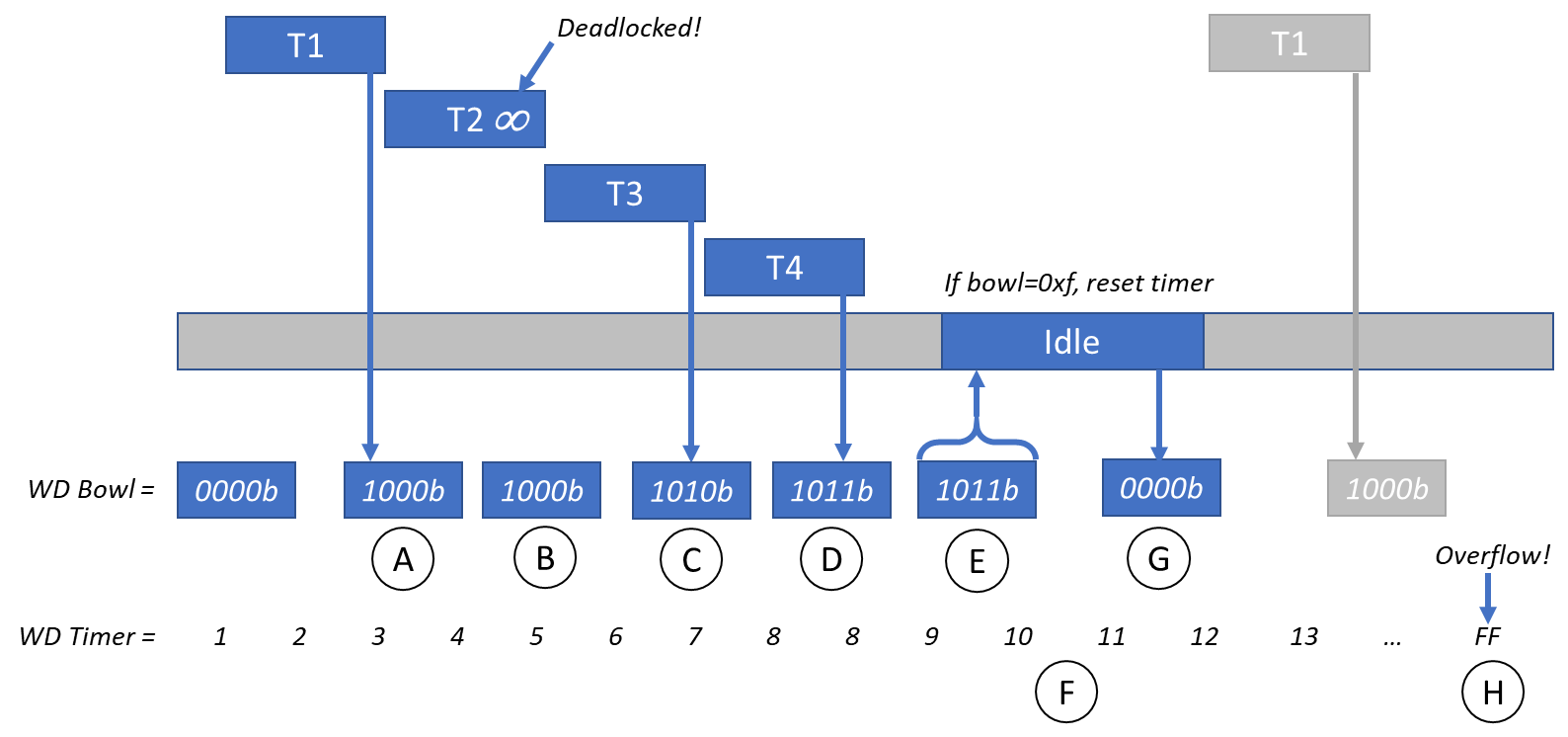
## Implementation

## Embedded System









## Web Service

# Testing

## Performance

# Accomplishments

## Objectives Met

## Limitations

## Lessons Learned

## Possible Improvements

# Comparison of Commercial Offerings

Several commercial IoT garage devices are available to consumers. These devices are typically developed by major opener manufacturers and marketed as an upgrade to the existing opener electronics. This section provides a short review of several of these offerings and their compatible hardware. Surprisingly, none of these technologies offered native integration with digital assistants such as Amazon Alexa. We provide a comparison between the GarageRTC features and the commercial offerings and discuss the benefits and drawbacks of these paired solutions.

## Commercial IoT Garage Interfaces

### Chamberlain MYQ-G0301

### Aladdin Connect

### Garage Door Buddy

### GoControl GD00z-4

## Comparison with GarageRTC Features

| **Feature** | MYQ-G301 | Aladdin Connect | Door Buddy | GoControl | GarageRTC |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Connectivity** |  |  |  |  |  |  |
| WiFi | ✔ | ✔ | ✔ |  | ✔ |  |
| Bluetooth |  |  |  |  | *1* |  |
| ZWave |  |  |  | ✔ |  |  |
| **Digital Assistant** |  |  |  |  |  |  |
| Alexa |  |  |  |  | *2* |  |
| Google |  |  |  |  | *3* |  |
| Siri |  |  |  |  | *3* |  |
| **Sense/Control** |  |  |  |  |  |  |
| Aux Control | *4* |  |  |  | ✔ |  |
| Amb. Temp |  |  |  |  | ✔ |  |
| Amb. CO |  |  |  |  | ✔ |  |
| Door Position | ✔ | ✔ | ✔ | ✔ | ✔ |  |
| **Local Control** |  |  |  |  |  |  |
| LCD Display |  |  |  |  | ✔ |  |
| Door | ✔ | ✔ | ✔ | ✔ | ✔ |  |
| Aux/Light |  |  |  |  | ✔ |  |
| **Open Source** |  |  |  |  |  |  |
| Web App |  |  |  |  | ✔ |  |
| FOSS |  |  |  |  | ✔ |  |
| Published API |  |  |  |  | ✔ |  |
| HW Schematic |  |  |  |  | ✔ |  |
| Universal |  |  | ✔ | ✔ | ✔ |  |
| **Reference** | [5] | [6] | [7] | [8] |  |  |
| 1. ESP32 contains the hardware but reference design does not implement.  2. Libraries exist that could integrate this capability. [9]  3. Can be integrated through webhook/REST API calls. [10]  4. Requires purchase of additional hardware. | | | | | | |

## How GarageRTC differs from current solutions

## Benefits and drawbacks of the integrated solution

The primary benefit of the commercially available devices is they have corporate backed development support. This translates to more refined software and user-friendly interfaces as they have superior developer resources as compared to the GarageRTC research team. They have the capacity to fully implement and test their solutions and maintain professional assistance hotlines. Further, mass production of their assemblies’ results in greater availability and prices typically ranging from $50 to $100. However, if the popularity of GarageRTC caught on, the resulting user community could quickly make up this gap. Further development could produce layouts to meet off the shelf housings or 3D printed enclosures.

The major drawbacks of the commercial units are their closed source nature. Vulnerabilities have been identified in the garage door solutions and countless other issues may exist. [11] These units have not undergone any sort of security community analysis and publish very little documentation on security controls available. The GarageRTC, however, doesn’t currently employ any security as it was not the focus of this effort, could easily have any flavor of security controls the developer prefers. Since the source is open, the security community could then review the implementation, identify potential flaws and make recommendations for remediation.

The closed source communication protocols also mean that the lifetime of these products is directly tied to the supplier’s willingness to continue support. If the companies go out of business or cease support in favor of a newer generation part, customers will be out of luck. The GarageRTC project offers the ability for anyone to stand up or share their own web application servers and therefore can indefinitely support hardware built to meet the GarageRTC protocols.

# Conclusions and Closing Remarks

# References

|  |  |
| --- | --- |
| [1] | A. Zeichick, "Throwing our IoT investment in the trash thanks to NetGear," *Network World,* 11 July 2016. |
| [2] | S. Levin, "Squeezed out: widely mocked startup Juicero is shutting down," *The Guardian,* 1 September 2017. |
| [3] | R. Suehle, "Q. What is the secret of Red Hat's Success?," *Technology Innovation Management Review,* January 2012. |
| [4] | R. Barry, "Mastering the FreeRTOS Real Time Kernal," Real Time Engineers Ltd, 2016. |
| [5] | Chamberlain, "MYQ Smart Garage Hub," Chamberlain, 2019. [Online]. Available: https://www.chamberlain.com/myq-smart-garage-hub/p/MYQ-G0301-D. [Accessed 13 April 2019]. |
| [6] | Genie, "Aladdin Connect," Genie, 2019. [Online]. Available: https://www.geniecompany.com/aladdinconnect/default.aspx. [Accessed 13 April 2019]. |
| [7] | GarageDoorBuddy, "GarageDoorBuddy - Remotely Control your garage doors from Anywhere," Garage Door Buddy, 2019. [Online]. Available: http://garagedoorbuddy.com/. [Accessed 13 April 2019]. |
| [8] | GoCOntrol, "GD00Z-4: Z-Wave Garage Door Opener Remote Controller Accessory," GoControl, 2019. [Online]. Available: http://www.gocontrol.com/detail.php?productId=4. [Accessed 13 April 2019]. |
| [9] | X. Pérez, "FauxmoESP - Amazon Alexa support for ESP8266 and ESP32 devices," 2019. |
| [10] | X. Pérez, "Using Google Assistant to control your ESP8266 devices," Tinkerman, 2019. [Online]. Available: https://tinkerman.cat/using-google-assistant-control-your-esp8266-devices/. [Accessed 13 April 2019]. |