TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

The purpose of this software design document is to describe the design of an Automatic Door Opener System.

1.2 Scope

The goal of the software running the Automatic Door Opener System is to control and monitor two motors and a push button.

1.3 Overview

The software design document is divided into 6 sections with multiple subsections. The sections are as follows:

1. Introduction
2. System Overview
3. System Architecture
4. Data Design
5. Component Design
6. Human Interface Design

1.4 Reference Material

- Memo titled *Automatic Door Opener Request Memo2 Revised12_07_18* given by Awesome Customer Inc.

1.5 Definitions and Acronyms

SBC	Single Board Computer
ADO	Automatic Door Opener

2. SYSTEM OVERVIEW

This is a system for controlling and monitoring the opening and closing of a door. The major components include 2 drive motors, 1 push button and the SBC. All data is collected and stored in the SBC. The SBC interacts with the push button by collecting data when the button is pressed. The SBC interacts with the 2 drive motors by sending data to the motors, telling them to either open or close. The SBC also collects data from the motors about the position of the door and state of the motor.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The complete functionality of the system consists of two subsystems, a control subsystem and a monitoring subsystem. The two subsystems collaborate via the SBC. The subsystems are responsible for commanding the motors to open or close when the push button is pressed and also monitor the status of the motors.

3.2 Decomposition Description

Classes to be used:

- Motor class to control and monitor the 2 drive motors
- Button class to manage the push button

3.3 Design Rationale

The motor class will be used to command the motor which will open or close the door. Member variables in this class will also be used to map to the essential bits in memory that are used by the motor.

The button class will be used to manage the button push/release action and maintain the current status of the button. It's member variables will be used to map the essential bits in memory that are used by the button.

4. DATA DESIGN

4.1 Data Description

Motor and Button class data is stored by reading the specific memory locations assigned to the push button and the two drive motors. This data is read and stored into member variables.

4.2 Data Dictionary

Motor Class

Member Variables:

- int motorAddress
- bool motorState
- int motorSpeed
- int motorPosition

Member Functions:

- int readFromMemory(int addr, int byteOffset, int bit)
- void writeToMemory(int addr, int byteOffset, int bit, int valueToWrite)
- bool isMotorMoving(int addr)
- int getMotorPosition(int addr)
- void openDoor(int addr)
- void closeDoor(int addr)
- void initializeMotorSpeed(int addr)
- double BinToDec(int[ ] arrayOfBits)

Button Class

Member Variables:

- int motorAddress
- bool buttonState

Member Functions:

- int readFromMemory(int addr, int byteOffset, int bit)
- bool getButtonState(int addr)

5. COMPONENT DESIGN

	Motor Class Member Functions
int readFromMemory(int addr, int byteOffset, int bit)	A method to read an address from a specific memory location. Returns the value in the memory location.

	int readFromMemory(int addr, int byteOffset, int bit) {
	
	}

void writeToMemory(int addr, int byteOffset, int bit, int valueToWrite)	A method to write to an address at a specific memory location. Returns the value in the memory location.

	void writeMemory(int addr, int byteOffset, int bit, int valueToWrite) {
	
	}

bool isMotorMoving(int addr)	A method to retrieve the state of the motor, whether moving or idle. Returns false on idle, true on moving.

	bool isMotorMoving(int addr) {
	value = readFromMemory(addr, 1, 3);
	return value;
	}

int getMotorPosition(int addr)	A method to retrieve the position of the motor. 0 = closed and1023 = open. Returns position as int.

	int getMotorPosition(int addr) {
	vector <int> bits;
	for (i = 4; i <= 13; i++) {
	bits.add(readFromMemory(addr, 1, i));
	}
	value = BinToDec(bits);
	return value;
	}

void openDoor(int addr)	A method to command the motor to open the door.

	void openDoor(int addr) {
	writeToMemory(addr, 2, 1, 1); //Command door to open
	bool opening = true;
	while (opening) {
	if (getMotorPosition(addr) == 1023) {
	writeToMemory(addr, 2, 2, 1);
	//Command door to stop
	writeToMemory(addr, 2, 1, 0);
	//Command to stop opening
	break;
	}
	}
	}

void closeDoor(int addr)	A method to command the motor to close the door.

	void closeDoor( ) {
	writeToMemory(addr, 2, 0, 1); //Command door to close
	bool closing = true;
	while (closing) {
	if (getMotorPosition(addr) == 0) {
	writeToMemory(addr, 2, 2, 1);
	//Command door to stop
	writeToMemory(addr, 2, 0, 0);
	//Command door to stop closing
	break;
	}
	}
	}

void initializeMotorSpeed(int addr)	A method to set the speed of the motor.

	void initializeMotorSpeed(int addr) {
	for (i = 3; i <= 5; i++) {
	writeToMemory(addr, 2, i, 1);
	}
	}

double BinToDec(int[ ] arrayOfBits)	A method to convert an array of binary values to a decimal. Returns decimal value.

	double BinToDec(int[ ] arrayOfBits) {
	
	}

	Button Class Member Functions
int readFromMemory(int addr, int byteOffset, int bit)	A method to read an address from a specific memory location. Returns the value in the memory location.

	int readFromMemory(int addr, int byteOffset, int bit) {
	
	}

bool getButtonState(int addr)	A method to retrieve the state of the button whether pressed or released. Returns 1 when released and 0 when pressed

	int getButtonState(int addr) {
	value = readFromMemory(addr, 1, 0);
	return value;
	}

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

N/A

6.2 Screen Images

N/A

6.3 Screen Objects and Actions

N/A