Design Implementation Of A Microcontroller Based External Facility Access Control System

2005

Thomas Fulbright
University of Central Florida

DESIGN IMPLEMENTATION OF A MICROCONTROLLER BASED EXTERNAL FACILITY ACCESS CONTROL SYSTEM

by

THOMAS E FULBRIGHT JR.
B.S. University of Central Florida, 2002

A thesis submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Engineering
in the Department of Electrical and Computer Engineering
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2005

ABSTRACT

In order to solve the College of Engineering and Computer Science facility access problem, an automated system that provides exterior doors with a time schedule and allows authorized users to gain access to the facility after hours was developed. A microcontroller based system has been designed to interface with a personal computer. The system designed within this thesis can be used as a starting point for multiple facility access control systems.

This thesis will describe the design, integration, test, and final delivery of a facility access system that incorporates the Texas Instruments MSP430 microcontroller, a magnetic card swipe reader, and software developed in Microsoft Visual Basic .Net to provide a reliable and robust system for the College of Engineering and Computers Sciences needs.

ACKNOWLEDGMENTS

I would to thank my parents for their support and love during the completion of my graduate studies. I am also very grateful to Dr. Samuel Richie for his guidance, support and advice for the completion of this work and my graduate studies.

I would also like to thank Usha Neupane for her enormous help and support in the completion of my work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE: INTRODUCTION

With the current number of faculty, students, and guest on campus everyday, building security has become a great concern. In recent years the College of Engineering and Computer Science has experienced several thefts that might have been prevented, if a facility access system was in place. Such an access system would control all exterior doors based on a schedule designed by facility management and grant access to users based on their credentials.

In today's market many access systems are available for commercial use. Such systems include the use of keyed doors, smart card technology, proximity readers and cards, magnetic card swipe technology, and many more. Currently, the College of Engineering and Computer Science has in place a contact smart card reader on all exterior and interior doors. This system was designed to allow users with the appropriate rights to gain access, unlock, and lock doors. Each user of the system is given a smart card that is programmed with the doors to which the user has access.

Currently, there are approximately 10 exterior doors with contact smart card readers in place that allow user access into the building. At the time of installation, the system provided the necessary facility access. As time progressed and parts began to wear out, the system could not provide the proper solution for the facility. Many of the exterior doors failed to lock and unlock correctly, or open on demand when a user tried to gain access after hours. Also, it was determined a time schedule was needed to secure exterior doors on a daily bases.

Other features that were available on the initial system install were tracking of users that requested access to unlock a door and the removal of user's access to individual doors. Both of these features became very useful, but also were cumbersome from its design. Facility

management physically has to go around to each smart card reader to retrieve the access list. The removal of users from doors is even more tedious. Again, facility management has to go to each smart card reader and verify the individual user does not have the right to gain access to the facility. Overall, this a time consuming process that could easily be automated.

For a new system to be productive, it must integrate seamless into the current system. A new system must provide a more efficient way of tracking users that wish to gain access to a facility, provide a daily schedule that would control when doors should be locked and unlocked. Also, this system needs to grow and expand with the changing needs of its clients.

1.1 Thesis Overview

This thesis will describe the design, integration, test, and final delivery of a facility access system that incorporates the Texas Instruments MSP430 Microcontroller, a magnetic card swipe reader, and software developed in Microsoft Visual Basic .Net. Chapter Two will describe the problem statement and design features of the system. Chapter Three will go into detail of all hardware used for the design. Chapter Four discusses the software developed for the microcontroller, and Chapter Five discusses the Host Software used in the system. Chapter Six will discuss the database design and Chapter Seven the implementation, non design work. Chapter Eight discusses testing and Chapter Nine is the conclusion.

CHAPTER TWO: PROBLEM STATEMENT

As stated in the Introduction, the current facility access system in place for the College of Engineering and Computer Science is in need of a slight modification. Many of the mechanical parts have stopped working correctly, a daily schedule of when the facility should be opened to the public is needed, and a way to track users that gain access to the facility after hours would help facility management in preventing theft of property. Theses modifications would allow the exterior doors to be fully automated.

2.1 Design and Features Overview

The facility access control system will be divided into two parts. The first part, the hardware design, interface, and integration, and the second part consist of the host software. The hardware placed at each external door will consist of one Texas Instruments MSP430 microcontroller connected to an external magnetic card swipe reader. The MSP430 will interface to PC via serial port communication to gain access to the facility schedule and send request for user access after hours. The host software will be split into a client/server application. A client will be considered the PC that the MSP430 interfaces with. The client will store the facility schedule and send periodical commands to the MSP430 to unlock or lock exterior doors. The server software will contain the master schedule for the facility and the list of users that can gain access to the facility after hours.

The system as a whole will consist of one door interface device at each exterior door, and a PC in a central location running the developed server software. This system will be able to provide such features as remote access control of all exterior doors. This will give administrators

the capability to unlock or lock doors at any give time. Also, administrators will be able to develop a facility access schedule that all exterior doors will adhere to. Finally, administrators will have a convenient way to see who is gaining access to the facility after hours and easily add and remove users that should have after hour access.

Figure two shows the block diagram of the door server. It consists of a computer and the software that will run on it. The door server will communicate with each door client via the Ethernet. TCP/IP protocols will be used to transmit information over the Internet. The door client will send periodic request for schedule updates that the server will issue a response to. Along with issuing schedule updates the door server will have the capability to unlock or lock any remote door at any given time.

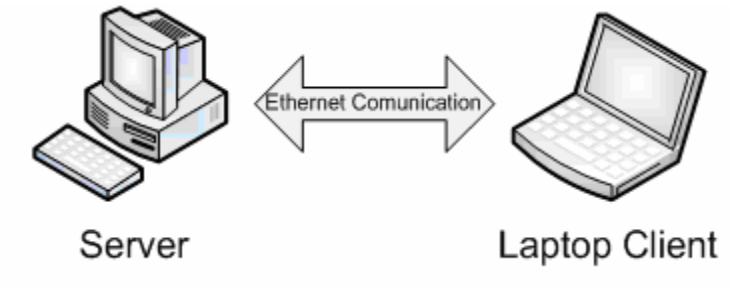

Figure 1: Door Server Block Diagram

CHAPTER THREE: HARDWARE DESIGN

This chapter outlines the design architecture, component selection and electrical schematic diagrams used to interface to the exterior doors. A special emphasis will be placed on Texas Instruments (TI) MSP430 microcontroller. Along with TI's MSP430, all other components used in the design will be described in detail.

3.1 Design Architecture

The door controller interface device consists of ten parts which are: TI's MSP430, MAX 3232 for RS232 interface, Quad High Current Peripheral Driver, four power relays, and Flash emulation tool port or JTAG port. The main component controlling all parts that interface to the exterior doors is TI's MSP430. This device directly interacts with a MAX232 chip for serial communication and a Quad High Current Peripheral Driver. Four relays are driven by the Quad High Current Peripheral Driver, which is driven by four digital outputs of the MSP430. Those four relays drive internal solenoids on the individual doors which allow them to lock and unlock. The block diagram in figure three shows the high level interaction between each component.

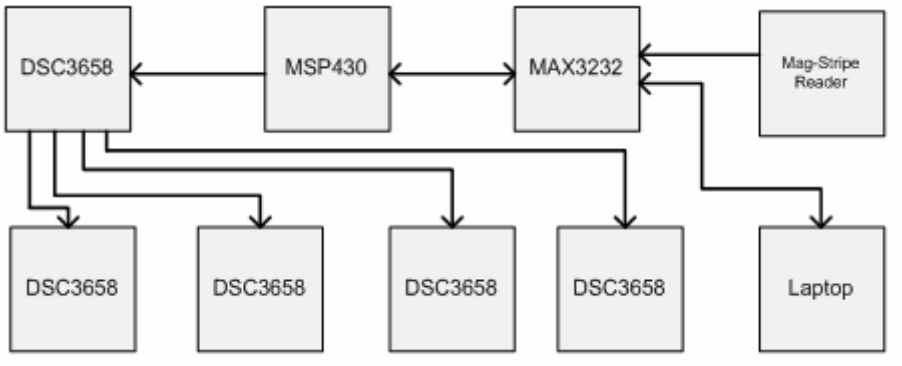

Figure 2: High Level Hardware Interface Design

The design of the system is quite simplistic. The MSP430 is used to relay signals from a laptop to unlock or lock the doors. Also, the MSP430 will take the magnetic card swipe input and forward that data to the laptop. That data is then compared to users in a database. If the user is found in the database a signal will be sent to unlock one door for 45 seconds. The challenge of this project was using the existing equipment and integrating into the current system.

3.2 Component Selection & Description

3.2.1 TI MSP430 Introduction

The MSP430 is designed as an ultra-low power microcontroller. It provides a modern 16-bit RISC CPU, peripherals, and flexible clock system are combined by using a von-Neumann common memory address bus (MAB) and memory data bus (MDB) [1]. The MSP430 also provides analog and digital peripherals interface. Partnering of a modern CPU with modular

memory-mapped analog and digital peripherals, the MSP430 offers solutions for demanding mixed-signal applications [2].

For the door controller interface, TI's MSP430F1232 will be used. It is a 28 pin surface mount chip. It's designed with 8KB + 256B flash memory and 256B RAM [3]. It provides one standard serial communication interface (UART). The MSP430 has a built-in 16 bit timer, 10 bit A/D converter with integrated reference and data transfer controller and fourteen or twenty-two I/O pins. Other key features of the MSP430 are listed on the next page.

Key Features of the MSP430 include:

- Ultra low-power architecture extends battery life
- High-performance analog ideal for precision measurement
- 16-bit RISC CPU enables new applications at a fraction of the code size
- In-system programmable flash permits flexible code changes, field upgrades and data logging.

3.2.1.1 TI MSP430 Clock System

The clock system used for the MSP430 was designed specifically for battery-powered applications. A low frequency auxiliary clock is driven directly from a common 32-kHz watch crystal [2]. This auxiliary clock can be used for a real-time self wake-up function. Also on the MSP430 is an integrated high-speed digitally controlled oscillator. This digital control oscillator can source the master clock used by the CPU and high-speed peripherals.

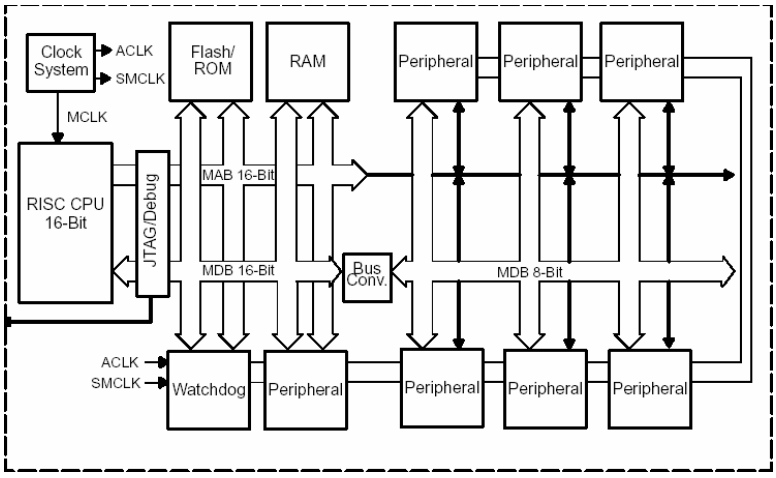

Figure 3: MSP430 Architecture [3]

3.2.1.2 TI MSP430 CPU & Instructions Set

As stated above the MSP430 uses a 16-bit RISC architecture. The CPU incorporates features specifically designed for modern programming techniques such as high level languages such as C. All operations, other than program-flow instructions, are performed as register operations in conjunction with seven addressing modes for source operand and four addressing

modes for destination operand [3]. Complementing the 16-bit RISC architecture are 16 registers. The 16 registers provide reduced instruction execution time. The operational time for register-to-register instruction execution takes once cycle of the CPU clock. Registers R0 to R3 are dedicated as program counters, stack pointer, status register, and constant generator. The remaining registers can be used a general purpose. As shown in figure four, peripherals are connected to the CPU using data address, and control buses. The MSP430 instruction set consists of 51 instructions with three formats and seven address modes [3]. Table one list the seven address modes and shows examples of how each is used.

Table 1: MSP430 Address Modes

Address Mode	**Example**	**Operation**
Register Mode	MOV R10, R11	R10 -> R11
Indexed Mode	MOV2(R5), 6(R6)	M(2+R5)-> M(6+R6)
Symbolic Mode		M(EDE)-> M(TONI)
Absolute Mode		M(MEM) ->M(TCDAT)
Indirect Register Mode	MOV @ R10 + R11	M(R10) -> M(tab + R6)
Indirect Auto Increment Mode	MOV @ R1010, R11	M(R10) -> R11 R10 +2 ->R10
Immediate Mode	MOV #45, TONI	#45 -> M(TONI)

Along with the MSP430 registers, it also contains RAM (random access memory), boot memory, information memory, code memory, and interrupt vectors. RAM is used for all global variables, scratchpad variables, and the stack. The boot memory contains the bootstrap loader. This is used for programming of flash blocks. Information memory acts as onboard EEPROM which allows for critical variables to be stored during power down. Code memory contains all code used to control the device. Interrupt vectors allow interrupts to occur from external inputs and will be discussed more in section 3.2.1.4.

3.2.1.3 TI MSP430 Operation Mode

The MSP430 was designed to have one active mode and five software selectable low-power modes of operation. The MSP430 can be awakened form any of the five lower power modes by an interrupt. Once finished processing the request the MSP430 can return to the lower power mode. This type of operation becomes very important for embedded devices and tools running off batteries. Table two describes each power mode.

Table 2: MSP430 Power Modes [3]

Power Mode	**Configuration**
Active mode AM	All clock are active
Low-power mode 0 (LPM0)	CPU is disabled ACLK and SMCLK remain active. MCLK is disabled
Low-power mode 1 (LPM0)	CPU is disabled ACLK and SMCLK remain active. MCLK is disabled DCO's dc-generator is disabled if DCO not used in active mode
Low-power mode 2 (LPM0)	CPU is disabled MCLK and SMCLK are disabled DCO's dc-generator remains enabled ACLK remains active
Low-power mode 3 (LPM0)	CPU is disabled MCLK and SMCLK are disabled DCO's dc-generator is disabled ACLK remains active
Low-power mode 4 (LPM0)	CPU is disabled ACLK is Disabled MCLK and SMCLK are disabled DCO's dc-generator is disabled Crystal oscillator is stopped

The design of the door controller interface will always use the active mode. A constant 3.3V will be provided at all times, thus allowing the system to stay in active mode. In future implementation and enhancement a battery backup or battery power should be considered to take advantage of the MSP430 low power consumption settings.

3.2.1.4 TI MSP430 Interrupts

The MSP430 was designed with multiple interrupts. All maskable interrupts are turned off by resetting of the GIE (global interrupt enable) flag in the status register [4]. Interrupts are used to control program flow based on events. The block diagram in figure five shows the how the interrupt process works on the MSP430.

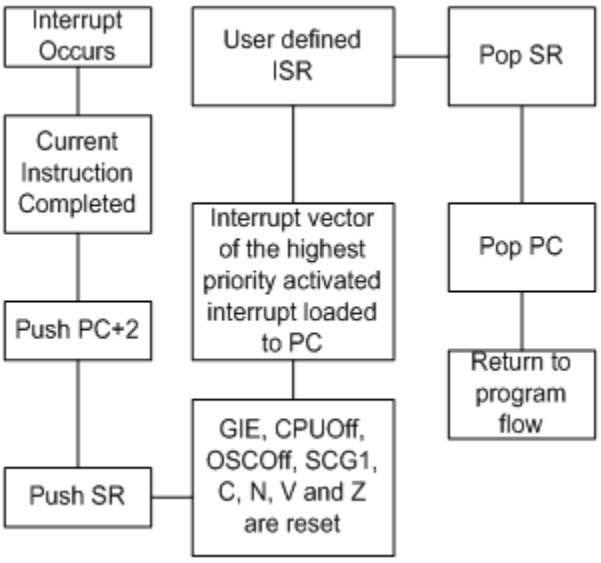

Figure 4: Interrupt Processing [4]

The MSP430 uses three standard types of interrupts: System reset, Non–maskable interrupt NMI, and Maskable. Non-Maskable interrupts are not masked by the general interrupt enable bit, but are enabled by individual interrupt enable bits [3]. Maskable interrupts are caused by peripherals with interrupt capability. System resets are generally caused by power resets. During System resets the status register is reset. All peripheral registers enter the power-up state. The interrupt latency is six cycles, starting with the acceptance of an interrupt request, and lasting until the start of execution of the first instruction of the interrupt-service routine [3]. The interrupt handling routine terminates with the instruction.

The MSP430 offers an alternative to interrupts by polling. Polling is the process of manually checking values for changes on a repetitive basis. Polling is performed in a loop of the main program. As with interrupts, polling can be dangerous. Since polling runs off the main programs loop, it can cause a normally short program to run extremely long and become time consuming.

3.2.1.5 TI MSP430 Inputs & Outputs

The MSP430 comes with three 8-bit I/O ports implemented on ports P1, P2, and P3. Each port has eight I/O pins. Each I/O pin is individually configurable for input or output direction. Also, each I/O pin can be individually read or written to. All I/O pins are controlled by memory-mapped registers [4]. Along with basic I/O functions, port pins can be individually configured as special as USARTs comparators signals, and ADCs. P1 and P2 are designed to be interruptible and P3 is non interruptible. In the door controller interface system, P1 pins will be used to control the individual doors and reprogramming of the chip, and P2 and P3 will be used for serial communication.

Non–interruptible ports take each bit as controllable. This allows inputs, outputs and dedicated function I/O to be mixed into a single port. Port pins are controlled by four byte-addressable registers. The first register is the direction register. The direction register controls the signal direction for port pins. The second register, the input register, is used to reflect the input value on the port. The third register, the output register, is used to write to output ports. The fourth register, the function select register, determines the individual pins on the I/O ports.

The interruptible I/O ports are very similar to the non-interruptible ports. Like the non-interruptible I/O ports the interruptible ports also contain the same control registers along with three addition byte-addressable registers. The first byte-addressable register is the interrupt enable. This register enables interrupts on individual pins. The second byte-addressable register is the interrupt edge select. This register is used to select the transition on which an interrupt occurs. The last byte-addressable register is the interrupt flag. This register is used to set a bit when an interrupt is generated.

All port registers can be changed in software. This allows software to control when interrupts are turned on and off. If port pins are configured as inputs, they will function as high impedance inputs. Interrupts on the MSP430 are edge triggered and susceptible to noise.

Incorporated in all the I/O pins is a standard USART, (universal synchronous/asynchronous receive transmit). The USART features 7 or 8 bit data with even, odd, or non-parity, separate transmit and receive buffer registers, built-in idle-line and address-bit communication protocols for multiprocessor systems. The UART character format consists of a start bit, seven or eight data bits, and even/odd/no parity bit, an address bit, and one or two stop bits [3]. Pins 3.4 and 3.5 of P3 on the MSP430x12x2 contain the standard USART. Pin 3.4 is used to transmit data to the MAX232 chip and pin 3.5 will receive data from the MAX232

chip. Also used, pin 2.2 of port 2 is configured to receive asynchronous data from the MAX232 chip which is generated by the magnetic card swipe reader.

3.2.1.6 TI MSP430 On-Chip Peripherals

The MSP430 was designed with several on chip devices. Those on chip devices are a hardware multiplier, comparator, LCD driver, and A/D converter. The MSP430 being used for the door controller interface only contains the A/D converter. Larger MSP430 have the built in hardware multiplier and comparator. The AD12 is a single 12-bit analog-to-digital converter with built in sample and hold, autoscan and data transfer controller. The ADC12 can achieve greater than 200 ksps maximum conversion rate [3]. The ADC12 implements a 12-bit SAR core, sample select control reference generator and a 16 word conversion-and-control buffer [3]. The ADC12 contains a multiplexer circuit that allows users to select one of eight external pins or one of four internal sources for the signal to convert. One useful tool built in on the MSP430is a temperature diode. Though not always accurate, it can provide useful low-cost alternative in certain applications.

The ADC12 uses four conversion modes. Those modes reflect the permutations of single and multiple conversions and one-time repeated conversions [4]. The four conversion modes are: single channel one-shot, single channel repeated multiple channels, single sequence and multiple channels repeated. As the name suggests, the single channel one-shot mode is a single conversion that stores the result in one of the ADCMEM registers. The single channel repeated, repeatedly performs conversion until stopped, again saving its result in the same ADCMEM register. The multiple channels, single sequence works very much like the single channel one shot. The multiple channels perform multiple conversions, looping through a specified number

of ADCMEM registers. Last the multiple channels repeated is identical to the single channel repeated, except the series of conversions is repeated until stopped.

3.2.1.7 TI MSP430 Interface

As shown in figure five, the MSP430 directly interfaces with three other components. The other components will be described later within the chapter.

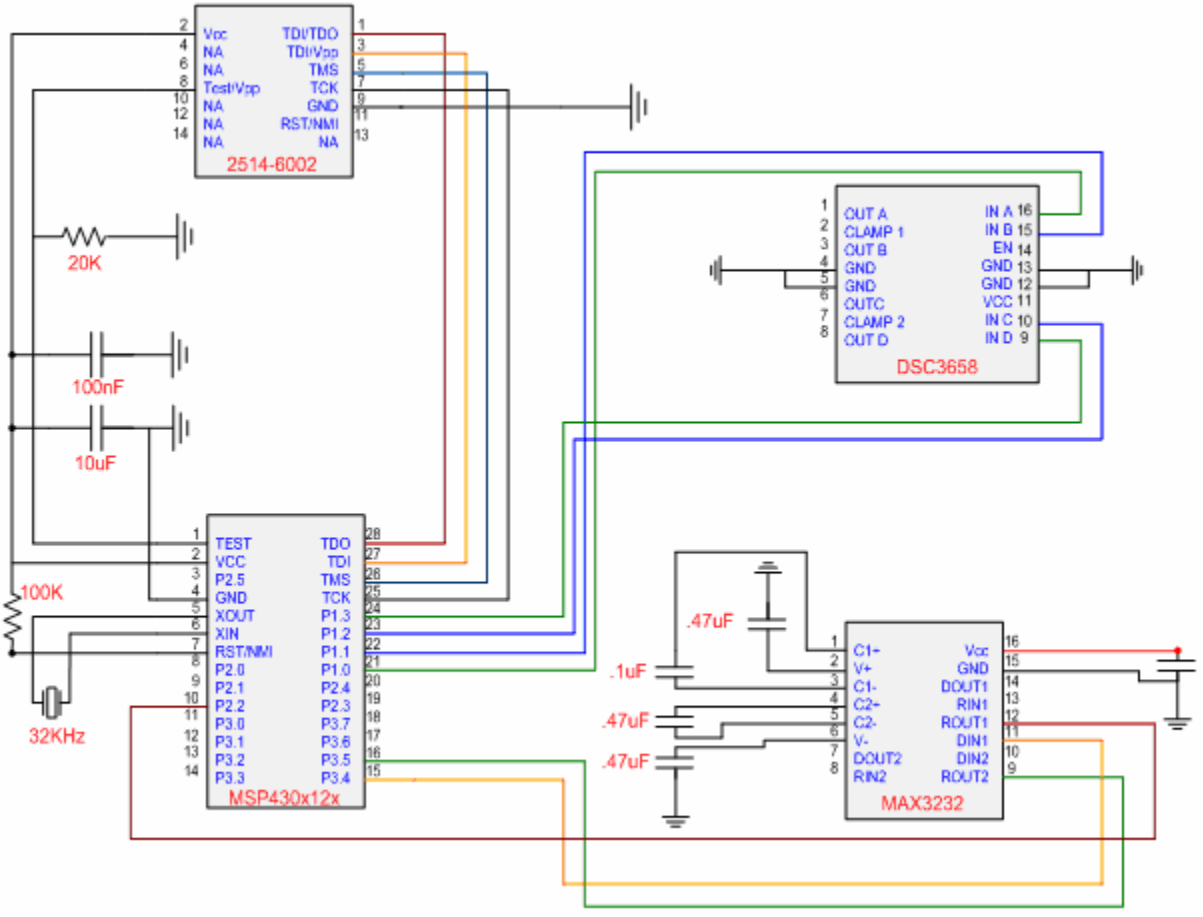

Figure 5: MSP430 Block Diagram Interface

3.2.2 TI MAX3232

To handle serial communication between the MSP430, magnetic card swipe reader, and the laptop computer, a MAX3232 chip by Texas Instruments will be used. Figure six shows the MAX3232 interface to the MSP430, magnetic card swipe reader and the DB9 port for serial communication with the laptop.

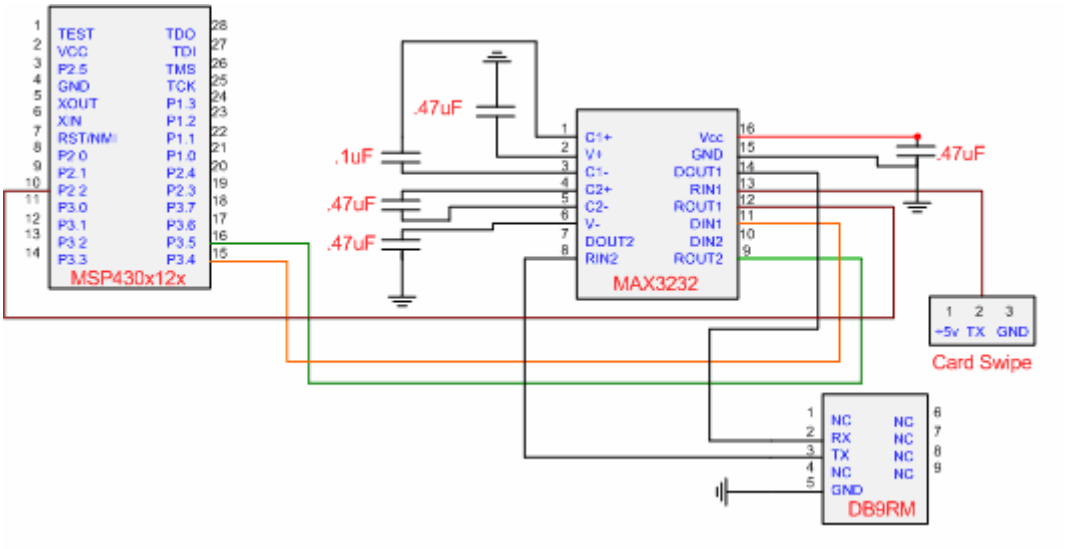

Figure 6: MAX3232 Interface diagram

The MAX3232 chip meets the requirements of TIA/EIA-232-F and ITU v.28 Standards. The TIA/EIA-232-F standard was introduced in 1962 in an effort to standardize the interface between DTE (data terminal equipment) and DCE (data communication equipment) [6]. It also operates with 3 V to 5.5 V V_{CC} and requires a low supply current of 300 µA typical [5]. The MAX3232 device consists of two line drivers, two line receivers, and a dual charge-pump

circuit. The MAX3232 operates at a data signaling rate up to 250 kbit/s and a maximum of 30-V/ µA driver output slew rate [5]. Table three shows the functions of each driver.

Table 3: MAX3232 Driver Tables

Each Driver

Input DIN	Output DOUT
L	H
H	L

Each Receiver

Input RIN	Output ROUT
L	H
H	L
Open	H

H = high level, L = low level, Open = input disconnected or connected driver off

The door controller interface system will take advantage of the MAX3232 dual drivers and receivers. Driver one will be used for communication from the MSP430 to the laptop computer. Receiver one will be used for data from the magnetic card swipe reader to the MSP430 and receiver two will be used for data from the laptop computer to the MSP430.

3.2.3 National Semiconductor DS3658

The DS3658 is Nation Semiconductors Quad High Current Peripheral Driver. It is designed for applications where low operating power, high breakdown voltage, high output current and low output ON voltage are required [7]. The DS3658 is the perfect choice to use in

the door controller interface. The MSP430 provides low output on voltage of 3V, and the high voltage needed to operate the doors solenoids is 17V. The DS3658 directly interfaces with the MSP430 and the onboard relays. Figure seven shows the interface of the MSP430 and four relays to the DS3658.

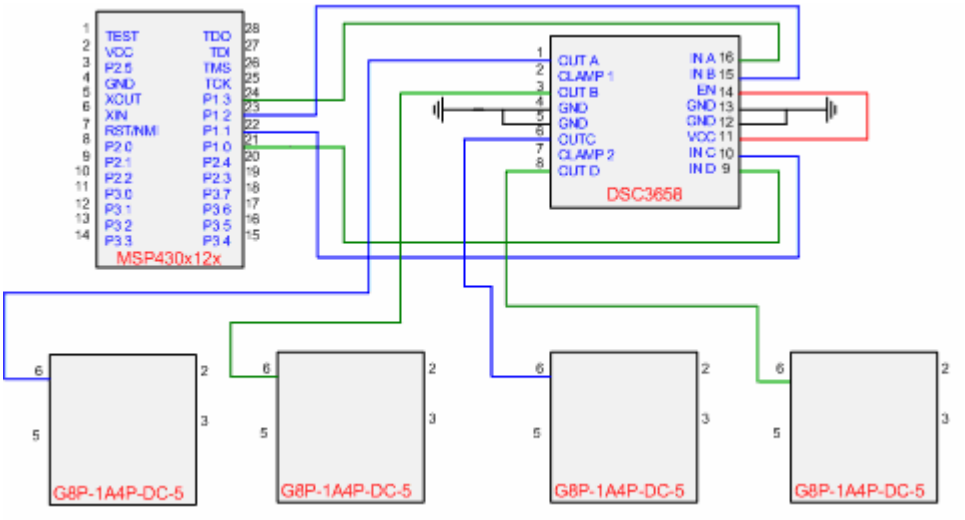

Figure 7: DSC3658 Interface Diagram

Some of the standard features of the DS3658 are: low standby power, high impedance TTL compatible inputs, low output ON voltage, output clamp diodes for inductive fly back protection and standard 5V power supply. The outputs of the DS3658 are capable of sinking 600 mA each and offer a 70V breakdown [7]. This range of sinking is well within the range of the onboard relays need for the door controller interface. Another useful feature of the DS3658 is its low standby power. The DS3658 typically runs off of 10 mW in standby mode. This feature can be usefully in future upgrades of the door controller interface where battery power might be used.

3.2.4 Omron Power PCB Relay

Driving the solenoid of each exterior door will be the G8P Omron PCB relay. The G8P is designed to handle up to 30 A of switching capacity. The G8P is available in several package styles, including open frame and sealed with ventable nib. For the door interface device, the sealed version was chosen over the open frame because of possible safety conditions. Currently each door system will be designed to contain four relays to operate four individual doors. If more than fours doors needs to be controlled the current design will not work. Figure eight shows the interface of the DSC3658 to each relay.

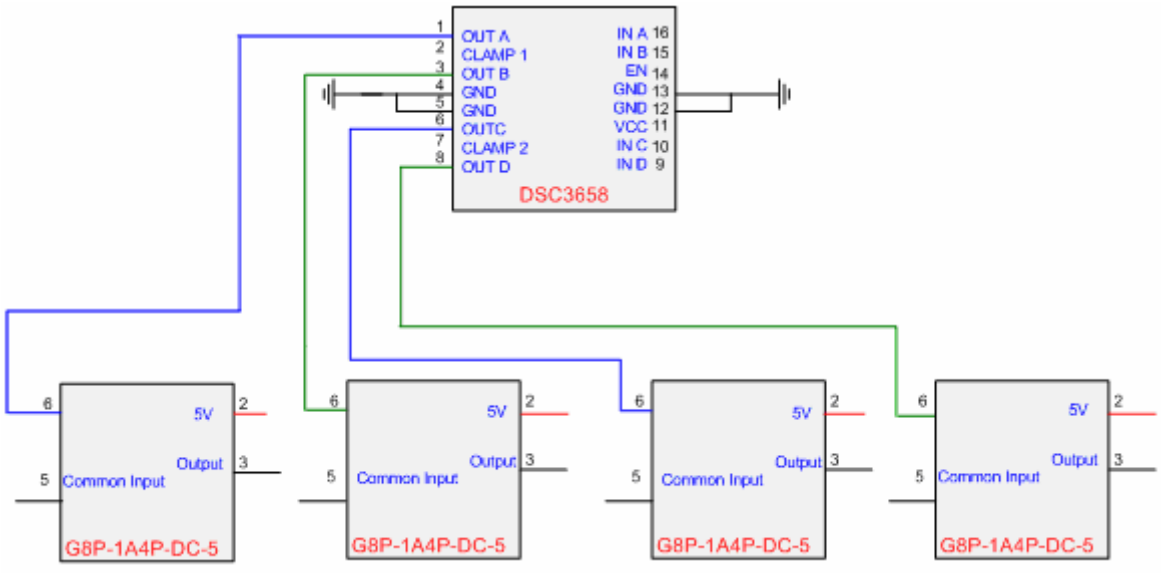

Figure 8: G8P Interface Diagram

3.2.5 Magnetic Card Swipe

At each entrance to the facility a magnetic card swipe reader will be placed. The magnetic card swipe reader selected for the door controller interface is Magtek port powered insertion reader. One of the main reasons for selecting this card reader is because of the modification that must be done to the already existing system. Also, the magnetic card swipe reader is designed to interface to a normal serial port which will have to be modified to integrate into the door controller system. Another advantage of using this magnetic card swipe reader is that it is port powered. Thus not require an external power supply.

As stated previous the magnetic card swipe reader is being used over the current smart card system. The current implementation of data on the smart card keys is encrypted when written to. This makes it very difficult to edit existing data and reverse engineering would not be ethical. Thus the use of the UCF ID card became a viable solution. Again the UCF ID card contains a smart chip, but this chip is being used to hold banking information, not allowing us to take advantage of it. However the magnetic stripe on the back of the UCF ID contains the needed information to determine if a user should have after hour's access to the facility.

The port powered smart card reader is capable of reading from tracks 1 and 2 off the magnetic stripe. Other design features of the magnetic card swipe reader include an open chassis design, ruggedized chassis and bezel material, ASCII message format at 9600 bps, and magnetic card swipe reading during insertion and removal of cards. Figure nine shows both sides of the magnetic card swipe reader.

Figure 9: Magnetic Card Reader

3.2.5.1 Magnetic Card Swipe Modes of Operation and Integration

The magnetic card swipe reader can operate in two different modes. The first is the buffered mode and the second is the unbuffered mode. The buffered mode works when a card is inserted and removed and a read is attempted. Once the card is removed the data is stored in the onboard memory buffer. This information is then transmitted when a request is received from the host. This data is stored in the local buffer until an erase command is issued by the host. For the door controller interface system, this mode will not be used. The unbuffered mode works when a card is inserted and removed and a read is attempted. If the read was successful the data is transmitted to the host. In this mode, there is no delay in sending data. Data is transmitted immediately after removing the card. The unbuffered mode will be used in the door controller interface system. The unbuffered mode allows the microcontroller to listen for new data at any given time. This stops the microcontroller from having to poll the reader looking for possible new data.

The integration of the magnetic card swipe reader includes interfacing it to the door controller interface PCB board. Normally this magnetic card swipe reader interfaces to a serial port on the back of a computer and all power and communication interface is done by the computer. For this project the jumper that provides power, ground and transmit data will have to be interfaced to the PCB board and the MSP430. Table four shows the jumper pins and their function.

Table 4: Magnetic Card Swipe Reader Interface Table

Pin Number	**Signal**
1	Transmit to PC
2	Receive from PC
3	+5 to +15 VDC
4	Ground

3.2.6 Power Supply

The power supply chosen for this project must provide power for the PCB board along with powering the doors solenoids. The power supply is a Cosel model UAW500s-24. It's designed to have a max output of 528W and a DC output of 24V at 22A [10]. The reason for choosing this power supply is based on the solenoid amperage draw. Each door requires 17A to engage the solenoid. The power supply currently in place is not capable of handling such large amperage draw.

CHAPTER FOUR: MICROCONTROLLER SOFTWARE DESIGN

This chapter outlines the design requirements, specifications, implementation, and integration of the microcontroller software. As any other microcontroller used in today's market, the MSP430 must be programmed to function correctly. When designing the MSP430, Texas Instruments made programming it very interactive. All code can be written in hexadecimal or C respectively. In the development kit TI provides the IAR assembler and complier used for programming the chip. The door controller interface system was programmed using IAR assembler.

4.1 Requirements

The main requirements for the door controller interface system are: interface with the individual doors, interface with the laptop computer and the magnetic card swipe reader. Each output controlling each door must work independently of other outputs controlling different doors. The UART on the MSP430 must be configured to use a configurable interrupt to send and receive data as needed to the laptop computer. Also, one of the digital I/O pins must be configured to receive data that is being sent from the magnetic card swipe reader. The full list of requirements is listed in section 4.1.1.

4.1.1 Specific Requirements

Table five goes over each requirement and its dependency, supporting material, and evaluation method.

Table 5: Specification of MSP430 Code

Requirement: 1
Requirement: Unlock individual door
Dependency: Connection to laptop, Laptop connection to server
Supporting Materials: Serial Cable
Evaluation Method: Send command to unlock each door individually, verifying each relay is driven open.
Requirement: 2
Requirement: Lock individual door
Dependency: Connection to laptop, Laptop connection to server
Supporting Materials: Serial Cable
Evaluation Method: Send command to lock each door individually, verifying each relay is driven closed
Requirement: 3
Requirement: Unlock all door
Dependency: Connection to laptop, Laptop receives command from server to unlock all doors
Supporting Materials: Serial Cable
Evaluation Method: Send command to unlock all doors. Verify all relays are driven open on a delayed sequence.
Requirement: 4
Requirement: Lock all door
Dependency: Connection to laptop, Laptop receives command from server to lock all doors
Supporting Materials: Serial Cable
Evaluation Method: Send command to lock all doors. Verify all relays are driven closed on a delayed sequence.
Requirement: 5
Requirement: Request entry
Dependency: Mag-stripe reader connected to board, Connection to laptop, Laptop sends data to server. Laptop receives response to unlock or not unlock doors
Supporting Materials: Serial Cable, cable from mag-stripe reader
Evaluation Method: Swipe card that is known to be in database and user has access to enter Verify one door is unlocked. Swipe card that is not know to be in database and user will not gain access

4.1.2 Interface Requirements

The following table shows the external input and output data to and from the system

1. The “Name” column gives the name of the data’s variable.
2. The “I/O” column contains either an “I” for input or an “O” for output.
3. The “Input/Output Format” column gives the format of the data as its input or output from the system.
4. The “Frequency” column gives how often the data will bin input or output from the system.

Table 6: MSP430 Interface Requirements Table

Name	**I/O**	**Input/Output Format**	**Frequency**
Card_Date	O	ASCII from card reader	Every card swipe
Door#o	I	ASCII from laptop	Every time laptop sends command to unlock. This can be adjusted based on configuration
Door#c	I	ASCII from laptop	Every time laptop sends command to lock. This can be adjusted based on configuration

4.2 Specifications

The software to be produced will provide a computerized interface between facility administrators and individual exterior doors. Specifically the MSP430 will accept commands from the client interface which will determine if doors should be unlocked or locked. Also, the MSP430 will send data from the magnetic card swipe reader to be processed. A high level architecture is shown in figure 10. The figure shows how a user will interface with the system and the system interface with the laptop.

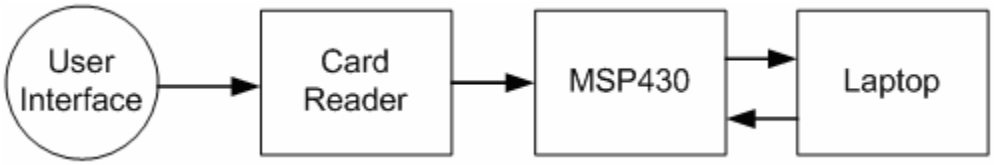

Figure 10: High Level Architecture

The MSP430 will rely heavily on the serial communication between the laptop and itself. The only user interface to the MSP430 will be the magnetic card swipe reader. This will be limited to after hour's requests to enter the building. If a communication break down happens between the MSP430 and laptop, the control of the exterior doors will not function correctly. This is a major risk with this project.

The Use case diagram in figure 11 shows a software overview of the functions that the MSP430 will perform.

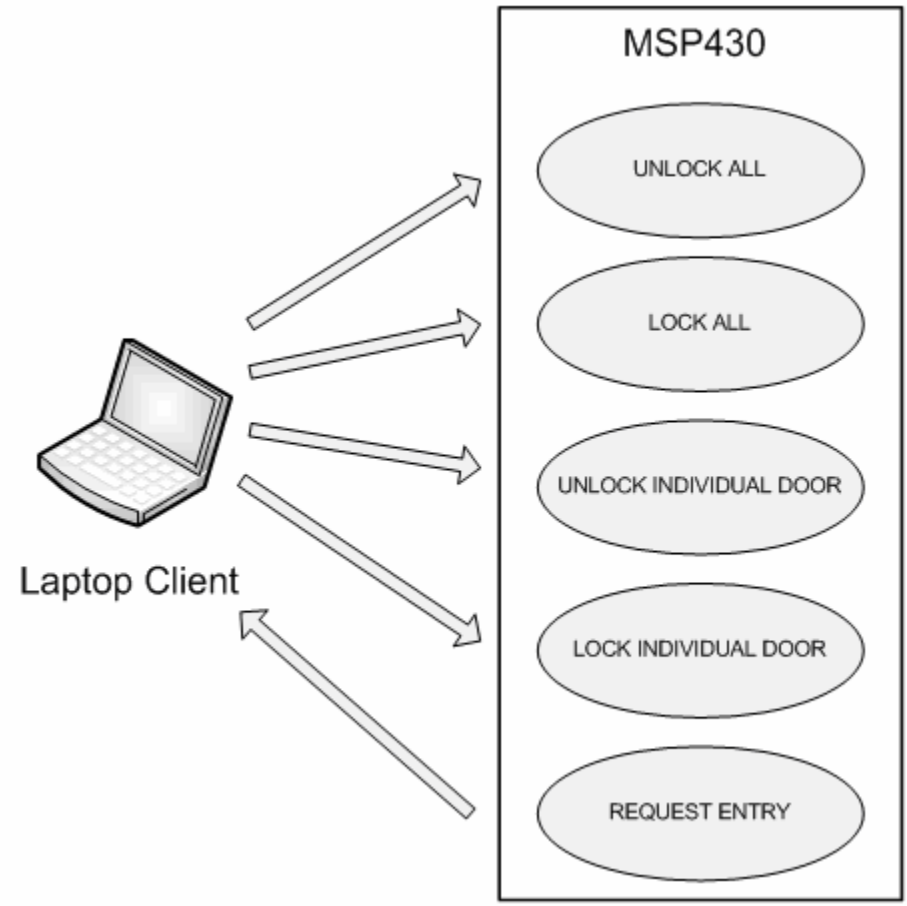

Figure 11: Use Case MSP430

Upon startup, the system should be ready to receive commands from the laptop and unlock or lock doors. If the MSP430 requires code changes, additional, or additional functions need to be added, or modification of the current code can be implemented by using the built in JTAG port.

4.3 Implementation

The implementation of the door controller interface system on the MSP430 will be accomplished in one file. The main file will contain several functions that will be used for communication between the MSP430 and laptop, controlling output pins, and interrupt control. The start of the program, unlike any other programming language, will begin in the Main function. Here all of the MSP430 setup will take place. Specifically the watchdog timer will be setup, the direction of I/O pins set, the transmit and receive speed of the UART set. As you can see from the code snippet below, almost all command in the Main function are specifically related to the MSP430 configuration.

```
WDTCTL = WDTPW + WDTHOLD;              // Stop watchdog timer
BCSCTL1 |= DIVA_3;                     // ACLK = LFXT1CLK/8
CCTL0 = OUT;                           // TXD Idle as Mark
CCTL2 = CM_1 + CCIS_1 + CAP + CCIE;    // CAP, ACLK, interrupt
TACTL = TASSEL_2 + MC_2;               // SMCLK, continuous mode
P2SEL = RXD;                           // P2.2/TA0 as RXD input

UCTL0 = CHAR;                    // UART0 CONTROL = 8-bit character...
UCTL0 = USART control register
UTCTL0 = SSEL0;// UART0 from aclk. UTCTL0 = transmit control register
UBR00 = 0x03;                    // 32k/2400 - 13.65
UBR10 = 0x00;                    //
UMCTL0 = 0x4A;                   // Modulation

P1DIR |=0xff;                    // P1.0 - P1.7 set to output
P1OUT &=0x00;                    // Set all P1 outputs off
ME2 |= UTXE0 + URXE0;            // Enable USART0 TXD/RXD
IE2 |= URXIE0;                   // Enabled USART0 RX interrupt
P3SEL |= 0x30;                   // P3.4,5 = USART0 TXD/RXD
P3DIR |= 0x10;                   // P3.4 output direction
_EINT();                         // Enabling interrupts
```

Figure 12: Main function MSP 430

The control of the output pins, specifically the control of the four pins that will drive the high power quad trip will be controlled by four functions. Each function will control one I/O pin that will either enable or disable the pin based on the command from the laptop. The next code block shows the functionality of how the MSP430 will enable or disable the I/O pins

```
void doorone(void)
{
   if (open_close == open)          // if open_close is equal to open
       P1OUT |= 0x01;               // turn pin 1.0 on
   if (open_close == close)
       P1OUT &= 0x0E;               // turn pin 1.0 off
}
```

Figure 13: Pin Enable MSP430

The communication with the laptop computer and magnetic card swipe reader is the last area of code to be looked at. In the main function of the program, a FOR loop is used to listen for data being sent to the MSP430. The other part of the communication is setting a normal I/O pin to emulate a UART pin by receiving the data from the magnetic card swipe reader. This is done by setting up an interrupt that reads from that selected pin. The code snippet in figure 13 shows one of the interrupts performing this function.

```
interrupt [TIMERA0_VECTOR] void Timer_A0 (void)
{
  CCR0 += Bitime;                       // Add Offset to CCR0

// RX
  if (CCTL0 & CCIS0)                    // RX on CCI0B?
  {
    if( CCTL0 & CAP )                 // Capture mode = start bit edge
    {
    CCTL0 &= ~ CAP;              // Switch from capture to compare mode
    CCR0 += Bitime_5;
    _BIC_SR_IRQ(SCG1 + SCG0);           // DCO reamins on after reti
    }
    else
    {
    RXTXData = RXTXData >> 1;
      if (CCTL0 & SCCI)           // Get bit waiting in receive latch
      RXTXData |= 0x80;
      BitCnt --;                        // All bits RXed?
      if ( BitCnt == 0)
//>>>>>>>>>> Decode of Received Byte Here<<<<<<<<<<<<<<<<<<<<<<<<<<<<
      {
      CCTL0 &= ~ CCIE;            // All bits RXed, disable interrupt
      _BIC_SR_IRQ(LPM3_bits);           // Clear LPM3 bits from 0(SR)
      }
//>>>>>>>>>> Decode of Received Byte Here <<<<<<<<<<<<<<<<<<<<<<<<<<<<
    }
  }
}
```

Figure 14: Interrupt of MSP430

4.4 Integration

Development and integration will be done using a personal computer and the IAR assembler and compiler provided with TI's prototyping kit. All code will be written in the compiler and debugged using the compilers tools. Figure 15 shows an example of the IAR compiler.

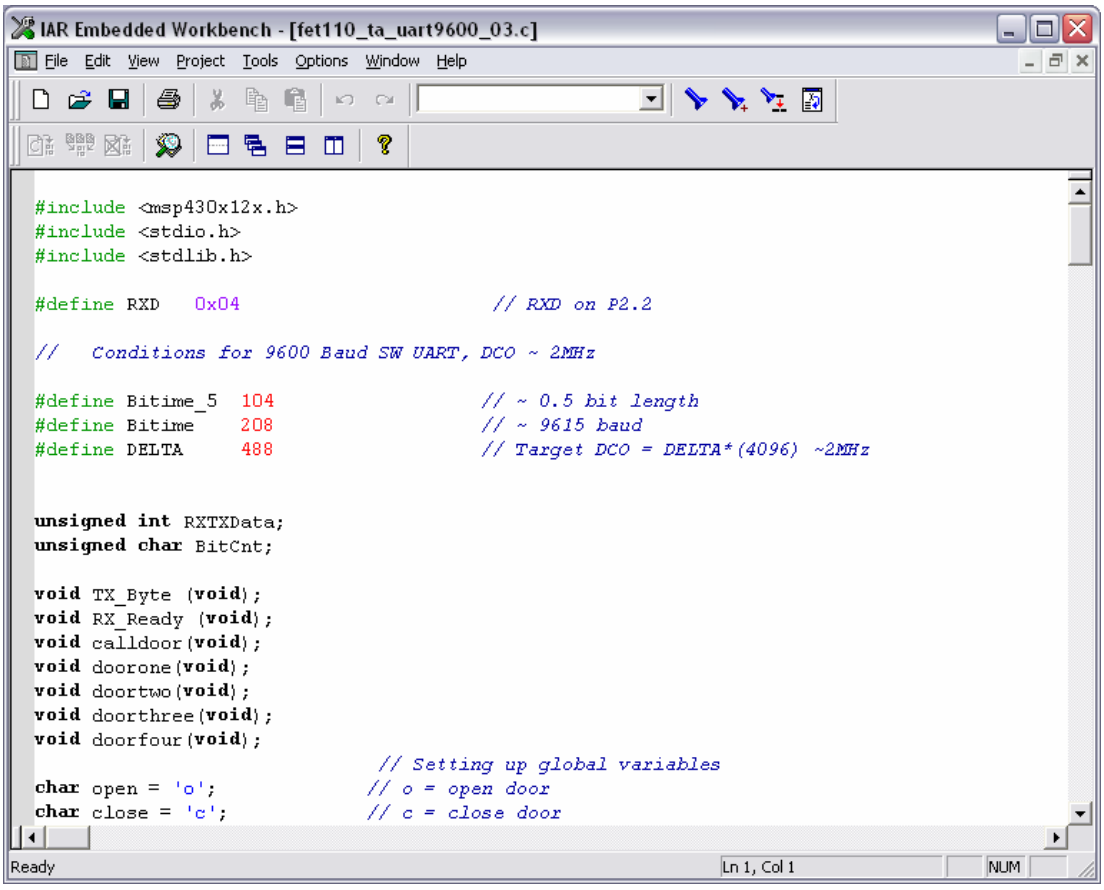

Figure 15: IAR Compiler

Like most compilers, the IAR compiler has the capability to add break points, add watches to variables, and print out the contents of memory of the MSP430. Once coding is complete, the code will then be loaded on the development board using the JTAG port. Then testing will take place verifying that each I/O pin is operating correctly. Also, communication using the UART port will be verified. Once testing concludes on the development board, The MSP430 will then be soldered on the production PCB board for full integration testing with all function parts. If any there are any additional problems, the production board contains a JTAG port for reprogramming of the MSP430.

CHAPTER FIVE: HOST SOFTWARE DESIGN

This chapter outlines the requirements, specification, software design diagrams, implementation and integration of the client/server software developed for use in the door controller interface system. Microsoft's Visual Basic .Net was chosen for development because of its easy prototyping abilities. Also, the amount of time to learn a new language would not have been advisable. All aspects of the software development process will be discussed.

5.1 Door Controller Client Software

Once a system is in place, a laptop computer will connect to the local Ethernet and the client application will begin. Figure 16 shows the client graphical user interface (GUI). The GUI is designed to be very simple but configurable by any user.

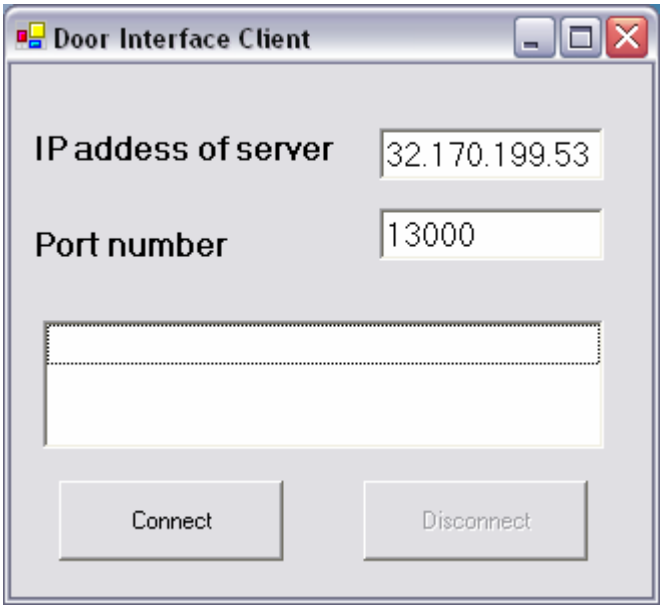

Figure 16: Client GUI

As you can see, there are two buttons, and text fields for the IP address of the server and port number of the server. Table 7 describes the functionality of each of the GUI components.

Table 7: Client Interface Functions.

Area or Button Name	**Function**
IP address of server	This provides the software with the IP address of the server when a connection is attempted
Port Number	This provides the software with the port number of the server that communication will take place on.
Connect	Sends an request to connect to the server
Disconnect	Notifies the server that the client is disconnecting

As stated earlier an Ethernet connection is needed for communication between client and server. In the center of the screen is a large text box that will display status messages sent from the server. Messages will only be displayed when the client is connected or a connection could not be made to the server. Table 8 shows the button pressed and the request sent to the server and the response from the server. Other communication takes place between the client and server for scheduling, request to gain access to the facility after hours and overrides of door states.

Table 8: Messages sent between client and server

Button Pressed	**Request sent to server**	**Response sent to Client**
Connect	Request connection to server	Connected to Door Server
Disconnect	Request disconnect from server	N/A

The table shows that once the connect button is pressed, an attempt to connect to the server will be made. Once the connection is made, the server responds back with a welcome message and the schedule for the client to unlock and lock the doors.

Figure 17 shows a high level overview of the client software. It consists of the GUI interface, the client application, RS232 interface, INI file interface and the connection to the server. Each part of the client software provides information to the client which is then passed on to the server. Specifically, the RS232 interface provides card data from a request to enter the building after hours and sends information to the MSP430 to unlock and lock doors. The INI interface is used to store configuration data about the client that is transmitted to the server. Each block will be discussed in more details later in the chapter.

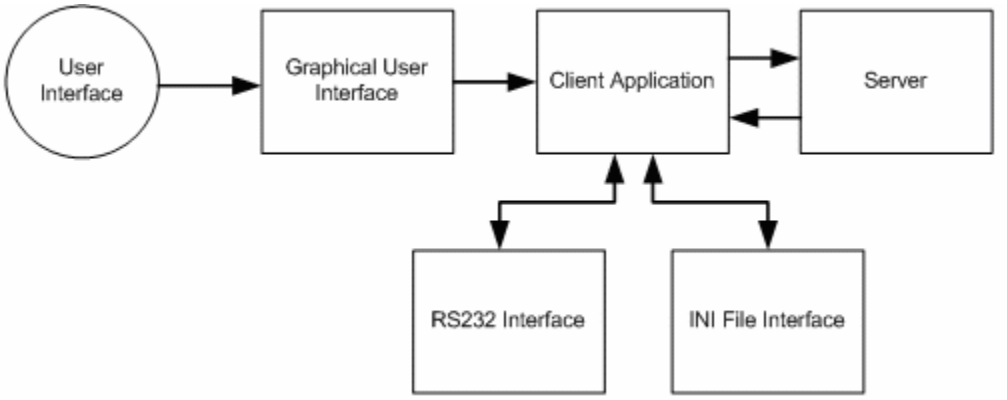

Figure 17: High Level Client Architecture

5.1.1 Client Software Implementation

For the client application an INI file will be used. This type of implementation was chosen over a local database at each client. The INI file, called info.ini, has multiple

configurable items in it. Each item is broken up into sections relating to the specific area for which the code will be used. The five sections contained in the INI file are: RS232, Server Address, Client Data, Schedule, and Door Value. Figure 18 shows an example of a typical INI file.

```
[RS232]
portNumber=1
baudrate=9600
databits=8

[Server Address]
IP=132.170.199.53
Port=13000

[Client Data]
location=CLIENT_MAIN_DOOR
NumDoors=4
DoorsState=UNLOCKED
Override=60000

[Schedule]
DAYofWeek=Tuesday
UnlockTime=8:00| AM
LockTime=7:25| PM

[Door Value]
Door1=LOCKED
Door2=LOCKED
Door3=LOCKED
Door4=LOCKED
```

Figure 18: Info.ini File Settings

The first section the RS232 is used to configure the client's communication settings with the MSP430. Like any other serial communication program, the port number, baud rated and data bits must be set. The second section, Server Address, is where the IP address and port

number can be entered for repeated use. The GUI on the client will automatically display the entered IP and port number but the user can change those values as needed. The third section, Client Data, contains configuration information about the physical door location. The location holds the door location. NumDoors holds the number of physical doors that will be connected to the door controller interface board. The DoorsState holds the condition of which the doors will be in once the system is powered up. The Override fields hold the milliseconds time of how long an override from the server will last. The fourth section, schedule, holds the current days schedule from the server. The last section, Door Value, holds the state of each door. This section is configurable based on the number of doors connected to the system. If less than four doors are used, then those doors can be removed from the list.

When the client loads its major task is to configure itself to connect to the server. It does this by first opening the INI file, reading the contents and storing the values. Once the values have been saved the GUI will be displayed and the client can then try to connect to the server. In Microsoft's Visual Basic .Net, the form load function calls several functions to initialize all components and set the global variables which hold the INI file parameters. Figure 19 shows the form load function.

```
Public Sub New()
  MyBase.New()

  'This call is required by the Windows Form Designer.
  InitializeComponent()
  iniInterface.SetGlobals()
  'Set text boxes with values from INI file
  txtPortNumber.Text = gblPortNumber
  txtIpAddress.Text = gblIpAddress
  tmrOverride.Interval = gblOverrideLength
  Add any initialization after the InitializeComponent() call
    End Sub
```

Figure 19: Client Form Load

Reading and writing from the INI file is accomplished by using the iniFile class written. The class uses the kerner32.dll to interact with the INI file. This class is designed to take in variable types and return that type. Figure 20 shows two examples of the variable types. The first example shows a request to read from the INI file and return a string. The second example shows a request to write a Boolean to the INI file.

```
'Get a string of text from a file
    Public Function GetString(ByVal Section As String, _
    ByVal Key As String, ByVal [Default] As String) As String
        ' Returns a string from your INI file
        Dim intCharCount As Integer
        Dim objResult As New System.Text.StringBuilder(256)
        intCharCount = GetPrivateProfileString(Section, Key, _
        [Default], objResult, objResult.Capacity, strFilename)
        If intCharCount > 0 Then GetString = _
        Left(objResult.ToString, intCharCount)
    End Function

' Write a boolean to the text file
    Public Sub WriteBoolean(ByVal Section As String, _
    ByVal Key As String, ByVal Value As Boolean)
        ' Writes a boolean to your INI file
        WriteString(Section, Key, CStr(CInt(Value)))
        Flush()
    End Sub
```

Figure 20: INI file interface functions

When each client is powered on a timer in the software is enabled. This timer is used to send updates to the microcontroller on whether the doors should be unlocked or locked. The RS232 class will be used to send those updated commands to the microcontroller. This class is designed to first open the communication port based on the INI settings. Figure 21 shows the function that opens the communication port.

```
Private Sub startComPort()

  csCommPort.Open(gblRs232PortNumber, gblRs232BaudRate,
                  gblRs232DataBits, Rs232.DataParity.Parity_None, _
                  Rs232.DataStopBit.StopBit_1, 4096)
  If csCommPort.IsOpen Then
     tmrReadCommPort.Enabled = True
  End If
End Sub 'Open com port
```

Figure 21: Opening RS232 on Client

Once the communication port has been opened, a timer is enabled to listen for any incoming data from the microcontroller. The only data sent from the microcontroller at this time, is the data off the UCF ID card. Once the data is sent, the client software will parse the data to put it in a format that the server will be able to read. Figure 22 shows the read port function along with the parser. Once the information is parsed and placed in the proper format the client will then send that data to the server.

```
Try
' As long as there is information, read one byte at a time and
' output it.
Dim strStream As String
Dim test As String
Dim strTrack1 As String
While (csCommPort.Read(1) <> -1)
  ' Write the output to the screen.
  test = Chr(csCommPort.InputStream(0))
  strStream = strStream & Chr(csCommPort.InputStream(0))
End While
If gblCardReaderID = "" Then
  gblCardReaderID = strStream
  strStream = ""
End If
If strStream <> "" And strStream.Length > 13 Then
  Dim intTotalLength As Integer
  Dim intTrack2Start As Integer
  Dim intTrack3Start As Integer
  Dim tmpParser As String
  'Creating a parser to remove " " from the input
  intTotalLength = strStream.Length
  intTrack2Start = strStream.IndexOf(" ^")
  strTrack1 = strStream.Remove(intTrack2Start, intTotalLength -
              intTrack2Start)
  strTrack1 = strTrack1.Trim
  strTrack1 = strTrack1.Remove(0, 2)
  strTrack1 = strTrack1.Replace("^", "|")
  strTrack1 = strTrack1.Replace("/", "|")
' now that we have removed the blank spaces we need to send the
  request to the server
  strStream = ""
  strStream = "|" & strTrack1
  writeToServer(strStream)
End If
Catch exc As Exception
 ' An exception is raised when there is no information to read.
 ' Don't do anything here, just let the exception go.
End Try
```

Figure 22: Reading Com Port on Client

Sending commands to the microcontroller is very simple. A text string is taken into the send function and then sent out to the com port. Figure 23 shows the code used to send commands to the microcontroller.

```
Private Sub SendCommand(ByVal strCommand As String)

  Try
    If csCommPort.IsOpen Then
       csCommPort.Write(strCommand)
    End If
  Catch ex As Exception
    MessageBox.Show("Error sending message to Micro Controller")
  End Try
End Sub 'Send command to microcontroller
```

Figure 23: Sending Commands on the Com Port of the Client

The other role the client has is the communication with the server. Communication is enabled by the user when they click on the connect button. When attempting to connect to the server the client uses the inputted IP address and port number. Once a connection is made the client then sends a message to the server letting it know its location and the number of doors at that location. The server will then responds with a connected message and shortly after send the schedule for that day of the times the doors should be unlocked.

The server interaction takes place in three main functions. The first function is the clientConnect function, the next function is the writeToServer function, and the last function is the ReadSocket function. The clientConnect function is used to actually connect to the server. The function uses the TcpClient and Stream class to establish the connection to the server and communicate with the server.

Figure 24 shows the connection function used by the client.

```
Private Function clientConnect() As Boolean

  Dim strServerMessage As String
  Dim counter As Integer

  Try
    tcpclnt = New TcpClient
    tcpclnt.Connect(txtIpAddress.Text, gblPortNumber)
    stm = tcpclnt.GetStream()

  Catch ex As Exception
    lstbCom.Items.Add("Unable to connect to the server at this time")
    cmdConnect.Enabled = True
    cmdDisconnect.Enabled = False
    Return False
    Exit Function
  End Try
  strServerMessage = gblClientLocation & "|" & gblNumDoors

  For counter = 1 To gblNumDoors
    strServerMessage = strServerMessage & "|" & gblInitialDoorState
  Next

  writeToServer(strServerMessage)
  tcpThd = New Thread(AddressOf ReadSocket)
  tcpThd.Start()
  Return True
End Function 'Connecting to server
```

Figure 24: Client Connection to Server

The ReadSocket and writeToServer functions are designed to use the TcpClient and Stream classes. The writeToServer class takes in a string that holds the command to send to the server. That string is then converted to ASCII format and transmitted using the stream write function. Error check is also used to verify that the stream is still connected to the server. If for some reason the client disconnects from the server, the client will handle this error by issuing a disconnect command to properly close the connection the client established. The ReadSocket function uses the stream connection to listen or read the stream. A read buffer is created that can

hold 100 bytes of data at a time. Once data is read into the buffer, those characters must be decoded in a form that can be used by the client. Then a compare is done on the command that was sent by the server. Table 9 list the commands and there function.

Table 9: Client/Server Commands

Command	**Function**
SERVER_DISCONNECT	Advises the client the server has closed its connection
UNLOCK_ALL	The server sends the request to unlock all doors
LOCK_ALL	The server sends the request to lock all doors
UNLOCK_TIME	The server send the unlock time for the day
LOCK_TIME	The server sends the lock time for the day
INDIVIDUAL_UNLOCK	The server sends the command to unlock one door
INDIVIDUAL_LOCK	The server sends the command to lock one door

5.2 Door Controller Server Software

The door controller server software is designed to operate on a windows based machine with the Microsoft's .Net Framework installed. The PC being used will be placed in an area that facility management will have easy access to it. The GUI designed for the server is shown in Figure 25.

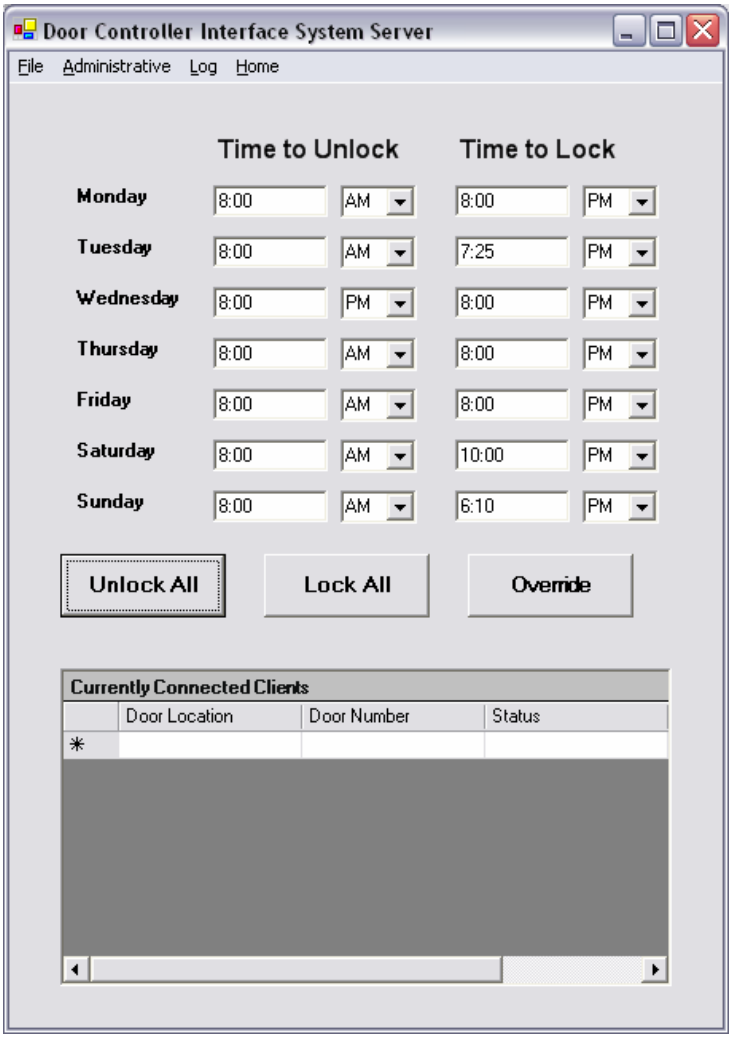

Figure 25: Door Controller Interface Server

The layout of the GUI allows users to easily configure a schedule of when the exterior doors should be unlocked and locked. The top half of the GUI displays the weeks schedule and the bottom half of the GUI displays the current door clients connected to the server and if the door is unlocked or locked. Table 10 describes the functionality of each of the GUI components.

Table 10: Server Interface Functions

Area or Button Name	**Function**
Time to Unlock	The first text box holds the time that doors should be unlocked. The drop down menu holds if the time should be AM or PM
Time to Lock	The second text box holds the time that doors should be locked. The drop down menu holds if the time should be AM or PM
Unlock All	Sends a command to all connected clients to override the current state and unlock all doors.
Lock All	Sends a command to all connected client to override the current state and lock all doors.

Other features that will be discussed in more detail later, are the ability to see a log of who has been given access to the facility after hours and who was denied access, administrative features to add or remove user access to the facility after hours, and the ability to unlock and lock individual doors.

Figure 26 shows a high level overview of the server software. It consists of the GUI interface, the server application, database interface, the actual server communication, and the log file writer. All server configuration settings are stored in the database. The server communication is done via the server class and the logging is done via the gblLogger class.

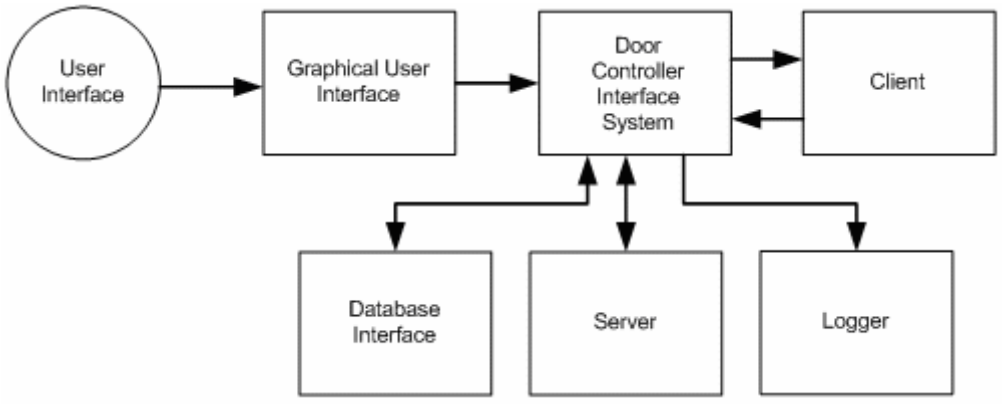

Figure 26: High Level Client Architecture

In figure 27, a flowchart describes the client server interaction based on the server first being started and a client connecting to the server. As the flowchart shows, the server will wait for a connection to the server then send a welcome message and schedule for the day. Once that information has been sent, the server will then listen for any request from the client. If a request is made, the server will process the request and send a response back to the client. Also, if a schedule change is made or command to unlock or lock the doors is selected, the server will send that change to the server. The server is configured to allow for multiple clients to be connected at any given time.

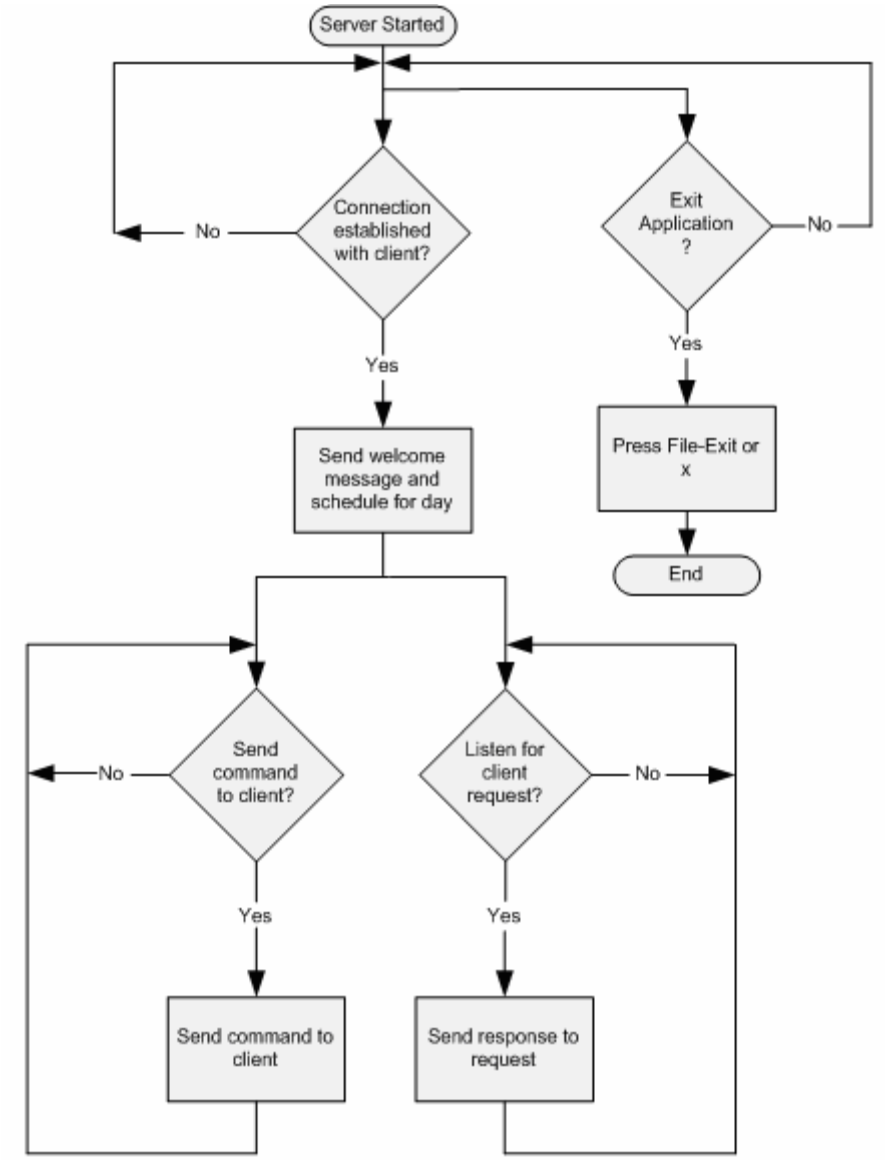

Figure 27: Client Server Interaction Flowchart

5.2.1 Server Software Implementation

The server application is configured to use a Microsoft Access database. The database was chosen because of the many configuration options on the server and also to keep track of users. When the software first starts a call to read the schedule from the database is performed.

Once the schedule is loaded, the GUI will be displayed. Once the GUI is loaded the user will have to enable the server to accept connections from the client. Unlike the client, the server saves its IP address and port number in the database. The call to start the server uses the TcpListener class which creates an open socket on the server. The server is designed to be multi-threaded to allow for multiple connections and multiple requests to be processed simultaneously. Figure 28 shows the StartServer function.

```
Public Function StartServer() As Boolean
  tcpLsn = New TcpListener(System.Net.IPAddress.Parse(gblIPAddress),
               gblPortNumber)
  tcpLsn.Start()
  tcpThd = New Thread(AddressOf WaitingForClient)
  tcpThd.Start()
nd Function
```

Figure 28: Opening socket on Server

The other major components of the server are reading the socket and writing to the socket. Writing to the socket is very similar to the clients write function. The one major difference between the two is the server has to know which client to send the message to. This is done by assigning each client an ID on connection to the server. The writeToClient function takes in the message to send to the client and the ID of the client. Then the message is encoded into ASCII format and sent to the client. Figure 29 show the writeToClient function.

```
Public Function writeToClient(ByVal strStream As String, ByVal realId
As Long)

  Dim encord As New System.Text.ASCIIEncoding
  Dim writeBuffer() As Byte
  Dim intLenght As Integer
  Dim clientData As ClientData = CType(dataHolder(realId), ClientData)

  Try
    writeBuffer = encord.GetBytes(strStream)
    intLenght = Len(strStream)
    clientData.structSocket.Send(writeBuffer, intLenght,
       SocketFlags.None)
  Catch ex As Exception
    'add logging
  End Try

End Function 'write To Client
```

Figure 29: Sending Commands from the Server to Clients

The readSocket function is much more complex than the clients read function. The server's readSocket function must first use the clients ID to determine where the request is coming from. The server next translates the message being sent from the client. After the translation is completed, the server will process the request which is shown in table 9 and send a response back to the client as needed. Depending on the request to the readSocket function will either make a database call or update the GUI for a client connection or disconnect. Figure 30 shows the all client request in code. Other functions in the server class include writing the lock times to the client, closing the thread when a client disconnects, and closing the server on shutdown.

```
' New welcome message from client
If (strStream.StartsWith("CLIENT")) Then
    strStream = strStream & "|" & realId
    ' check to see if door location was found in database
    dbInterface.overrideLookup(strStream)
    FrmM.AddtoTable(strStream)
    writeToClient("Connected to Door Server", realId)
    'send the current unlock and lock time
    writeLockTimes(realId, False)

'Client request to unlock a door
ElseIf (strStream.StartsWith("|")) Then
    blnAllowAccess = dbInterface.dbRequestLookup(strStream)
    If blnAllowAccess Then
       writeToClient(gblReqtoUnloc & "|1|Unlock", realId)
    End If

' Client disconnect
ElseIf (strStream.StartsWith("DISCONNECT")) Then
    FrmM.removeClientFromTable(realId)
    CloseTheThread(realId)

'Update table to show all doors are locked
ElseIf (strStream = gblLocAll) Then
    FrmM.updateTable("LOCKED", "Status", realId)

'Update table to show all doors are unlocked
ElseIf (strStream = gblUnlAll) Then
    FrmM.updateTable("UNLOCKED", "Status", realId)
End If
```

Figure 30: Server Read Socket Function

The part of the server that helps with configuration and dealing with client request is the database interfaces. The database interfaces is used whenever the server application is first started, a change is made to the GUI by the administrator, a request for access from the client and many other features. On startup, the server application makes a call to open an ADODB connection with the access database. Once that connection is made all configurable parameters are loaded and their respective variables are set. Next the call to populate the weekly schedule is made. This is accomplished by creating a query to select all information from the table. That

information is then returned to the main form of the server to populate the GUI. Figure 31 shows the readSchedule function.

```
Public Function readSchedule() As ADODB.Recordset
  Dim strQuery As String
  Dim recordset As ADODB.Recordset

  recordset = New ADODB.Recordset
  strQuery = "Select * from Door_Schedule"

  Try
    recordset.Open(strQuery, localDB)
  Catch ex As Exception
    gblLogger.gblLogMessage("dbInterface:readSchedule", "Error
        attempting to read from the database")
  End Try

  If Not recordset.EOF Then
    Return recordset
  End If
End Function ' Read the schedule from the database
```

Figure 31: Server Database Request for Schedule

Another role of the database interface is the request lookup function. This function is used to process a request for entry into the building from the client. This function takes in the UCF ID card information and searches the database for that user. If the user is found a boolean variable is set to true and the client is given access to allow the user entry. Also, during this time the users information is also being stored in the database along with the time they requested access and if access was granted or not. Another function in the database interface class is the update schedule function. This function is used when any part of the schedule has changed. The function takes in the field name that changed the fields new value and the day of the week of the change. With that information a query is created and the database is updated. Other functions in

the database interface class include the database closer, database configure parameter, read door table names, get number of doors, get unlock time, and get lock time.

An important class of the server application is the Logger class. This class is designed to log all events of importance that take place on the server. This class is referenced by all other classes within the application. The class is first used when the application is first started. A call is made to create a new log file with current date and time appended to the front of the log file name. When a call is made to add information to the log file the function gblLogMessage is called. This function takes in the parameters of where the call is being made from within code and the message to write to the log file. Figure 32 shows the gblLogMessage.

```
Public Function gblLogMessage(ByVal strModule As String, ByVal
                strMessage As String)
  Dim strMsgText As String

  strMsgText = Format(Now(), "MM/dd/yyyy HH:mm:ss - ") & "(" &
                strModule & "): " & strMessage
  Try
    glbLogFileHandle = File.AppendText(gblLogFileName)
    glbLogFileHandle.WriteLine(strMsgText)
    glbLogFileHandle.Close()
  Catch ex As IOException
    MessageBox.Show("Could not write to log file.")
   End Try
End Function
```

Figure 32: Server Logger Function

Another log that is available on the application is the Facility Access Log. This log displays the requestor UCF ID card number, the time they requested access and if access was granted. This information is recorded every time a user attempts to gain access to the facility after hours. This feature can help facility management determine people that our trying to gain

access to the facility after hour's but are not allowed. Also, this tool could also help define a more appropriate schedule if the facility is being accessed more during non open times.

In designing the server software, it was determined that the administrator would need to be able to view, add and delete users that could gain access to the facility after hours. All of these features are done in the Administrator pull down menu. The list of active users that are allowed to enter the facility after hours displays the users first name, last name and the users UCF ID card number. This information is intended to be populated at random intervals, such as each semester. The individual add feature is useful if access needs to be granted to a new user in between data reload. The administrator simply needs to choose add from the menu system and enter the users first name, last name and UCF ID card number. This feature was designed to be simple and straight forward. To remove a user, again the administrator needs to enter the users first name, last name and UCF ID card number. Then click on the remove button and the user will be removed from the facility access list. Figure 33 shows the Add User and Remove User screen.

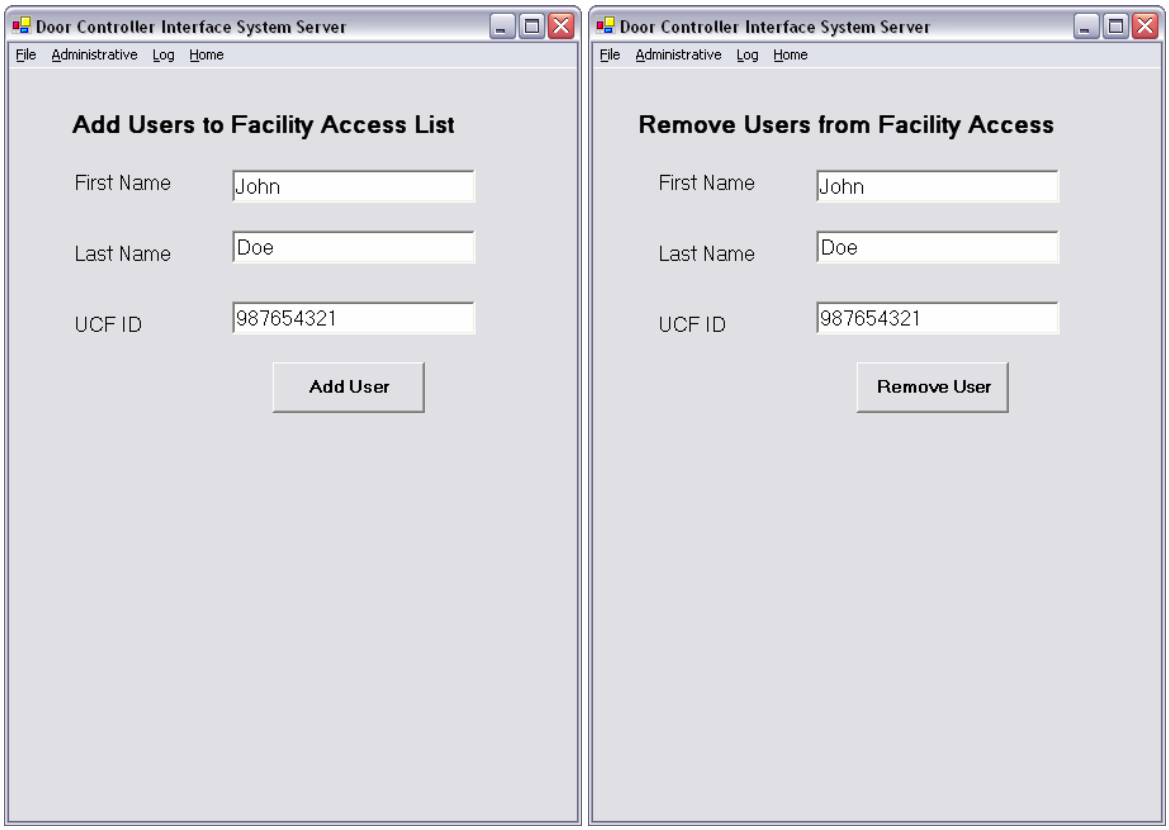

Figure 33: Add Remove Users

5.3 Software Life Cycle

The software lifecycle model chosen to follow, serves as a cross between the waterfall model and the rapid prototype model. First, requirements are determined and reviewed. Next the specifications for the product are documented and it clearly states what the product is to do. When all terms have been agreed on, the design phase is started. In parallel with the first two phases, a rapid prototype is created to gain a better understanding of the proposed project. The rapid prototype serves as a base for the ultimate design. When the design is completed, the implementation phase begins. In a normal process, the implementation and integration phases

will happen in parallel. The last phase of software life cycle is testing. This phase will thoroughly test all functionality of the software to make sure all specifications and requirements are met. The feedback loops at each phase allow for the proper modifications. It should also be noted that in this type of development, it is not likely all things will work correctly the first time.

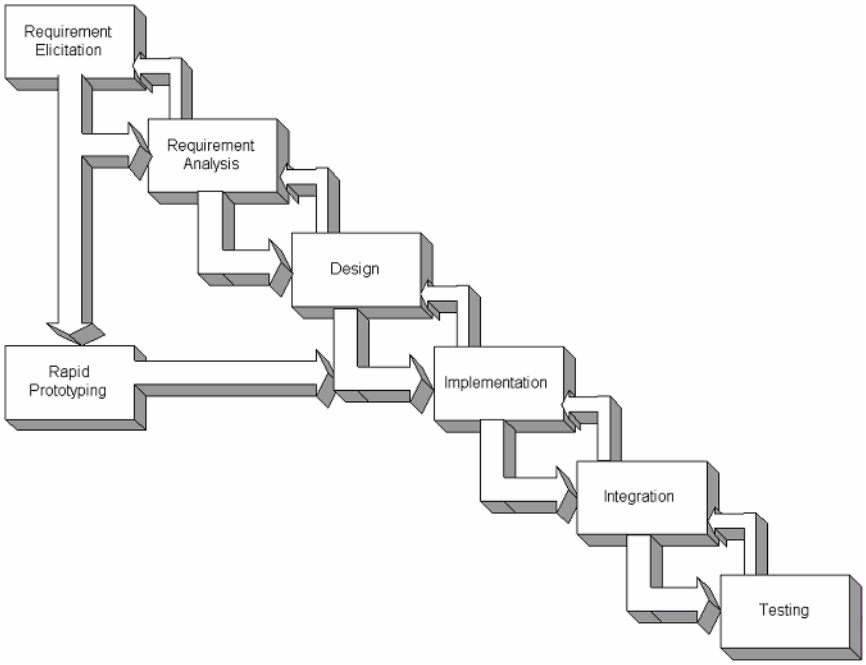

Figure 34: Software Lifecycle Model

CHAPTER SIX: DATABASE DESIGN

This chapter discusses the development and design of the database used in the server application. The back bone of the server application is the Microsoft Access database. The Access database was chosen because of easy integration into the Microsoft .Net environment and also being user friendly. Another reason for choosing the Access database is it only requires one file to run it. Many other databases require the installation of support files for the database to work correctly. In designing the system, it is believed that any database could have been used. Some reasons for not choosing other databases include, cost, unfamiliarity, speed, and computer speed.

6.1 Table Descriptions

The design of the tables for the door controller database is implemented in such a way that allows them to have expandability and flexibility. It is expected that the system will continue to grow with future modifications. The database is made up of six tables. Each table contains information related to users or configurable parameters needed for configuring of the server correctly.

The first table designed was the system configuration table. This table holds all value that will be used in configuring the server application. The table is designed with four columns: Item Name, Current Value, Default Value, and Description. The item name column holds the key word of the value being stored. An example is the IP address of the server. The current value column contains the current value for the key word. The Default value holds the first

configured value of the key word. The description column holds the description of what the key word is used for and the possible values it should have.

The next table is the door schedule table. The door schedule table holds the weekly schedule of when all exterior doors should be locked and unlocked. This table is designed with five columns: Day of Week, Time Unlock, Time Unlock AM PM, Time Lock, and Time Lock AM PM. The Day of Week column holds the day names, Monday – Sunday for the week. The Time Unlock and Time Lock columns hold the numerical number when the exterior doors should be unlocked. The Time Unlock AM PM and Time Lock AM PM columns hold a zero or one relating to AM or PM. The default settings for this table are to have all doors unlock at 8:00 AM and locked at 8:00 PM.

The next table is the door table. This table holds the location of the clients that have connected to the server and the number of doors at that location. This table is designed with two columns: Door Group Name and Number of Doors. This table is updated every time a new client connects to the server with a new door location. This table is used for tracking of all new clients that connect.

The next table is the door access list. This table is used to hold all users that will have access to enter the facility after hours. This table once populated could grow to hold over 4,000 users or more. This table is designed with three columns: UCF Card ID, First Name, Last Name. The UCF Card ID holds the users card number. The First Name column holds the first name of the user. The Last Name Column holds the last name of the user. Other columns considered during the designing of this table are social security number, major, age, and address. It was determined that the UCF Card ID would provide enough information to gather more data on a particular user.

The next table is the Administrators Table. This table is used to hold all administrators that will have access to the server software. The table contains four columns: First Name, Last Name, User Name and Password. The First Name column holds the administrator's first name. The Last Name column holds the administrator's last name. The User Name holds the administrator's login user name. The Password field holds the administrator's password to login into the system with.

The last table is the Access List table. This table stores each request for access to the facility when a card is swiped at the exterior door location. The table contains three columns: UCF ID, Time Accessed, and Granted. The UCF ID column holds the card number of the person trying to gain access to the facility. The Time Accessed contains the date and time the person requested access to the facility. The Granted column contains the word true or false. If true is found in the column that means the user was granted access to the facility. Like the door access table the access list table could continue to grow very large.

As requirements change, the tables in the database can be modified very easily to allow for requirement changes. The System Configuration table could continue to grow, if more features are added to the server software. In time, if the system grows to include interior doors, other tables might be needed.

CHAPTER SEVEN: IMPLEMENTATION – NON DESIGN

This chapter discusses the implementation of the building and connection of all the components and software together. Specifically, the design of the PCB board, Milling of the PCB Board, component mounting and future fixture enclosure will be discussed.

7.1 PCB Design

Once all hardware components were selected, a custom PCB had to be made. To make the PCB board, Altium PCAD 2002 was used. This software provides all the necessary features to design each individual component and transpose the electrical wiring schematic to the PCB board. Figure 35 shows a screen shot of Altium PCAD 2002. The current display of the PCB board is displayed on the screen.

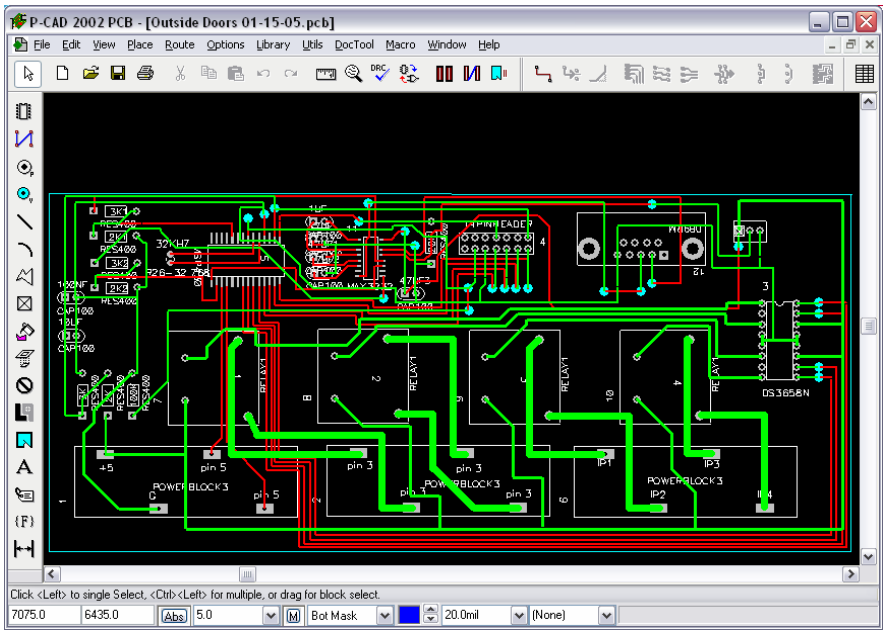

Figure 35: Altium PCAD 2002

As stated above, each component was hand designed in PCAD2002 to provide a complete library for the design. The design of each component was done using the included pattern editor and referencing the data sheets provided for each component. Once each pattern was designed, the layout of the board began. It was determined that the wiring block should have easy access to the outside of the PCB board. Given the size of the wiring block and the requirement of little interface the best location was on the edge of the board. The next component placed was the relays. After physically moving the relays around on a board template, it was found they fit best lined up in a row next to the wiring blocks. The last few components, the MSP430, MAX3232, DS3658, RS232 port, and JTAG port were positioned in the remaining space on the board. Next the necessary resistors and capacitors were placed around their respective components. The last part of the board design was trace width. A minimum trace width of 20 Mils was chosen. For connection with larger currents, a trace width of 70 Mils was used. The larger traces include connections between the relays and wiring blocks. Once all connections are completed, the board layout is transferred to the milling machine software.

7.2 PCB Milling

Once the PCB board design is finished in the PCAD 2002 software, the file is then transferred to the ISOPRO software used to run the milling machine. Before the board can be milled, more work must be completed on the design in ISOPRO. First all holes are placed where a thru hole must be drilled. Next isolation layers must be placed on the design. The isolation layer is the actual copper that will be removed from the board leaving the copper trace. Figure

36 shows the Quick Circuit Milling Machine. The final thing that is done before milling is applying a rubout layer. The rubout layer removes all copper on the board that is not being used. This helps prevent the possibility of shorts.

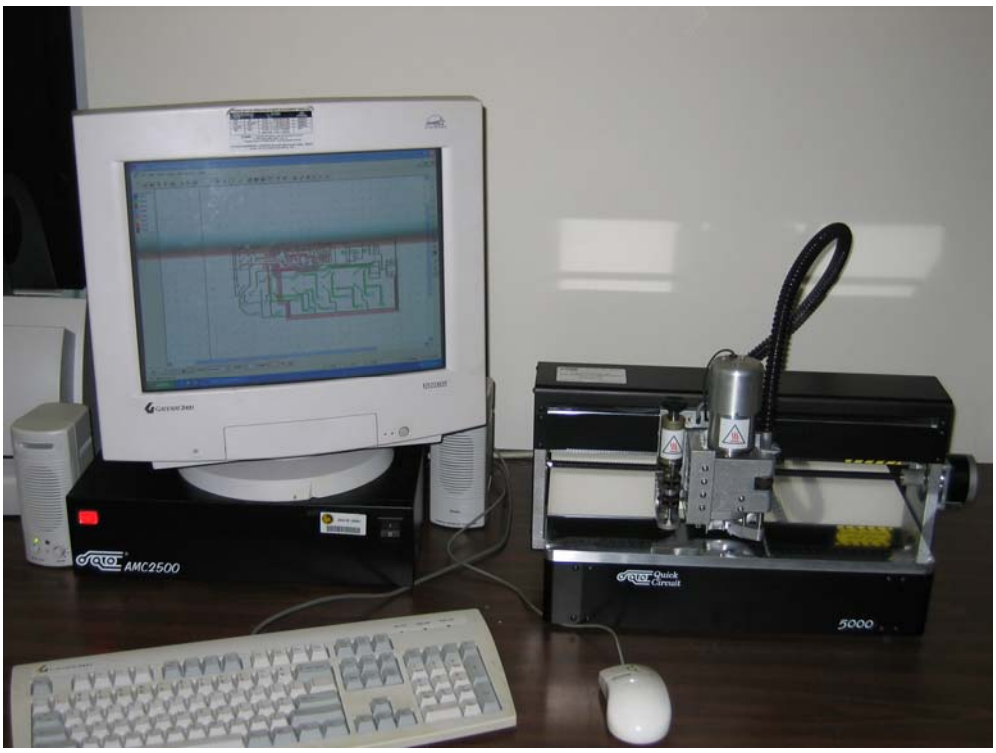

Figure 36: Quick Circuit Milling Machine

Figure 37 contains two photos of the milling machine at work. The picture on the left shows the milling of the isolation layer and the picture on the right shows the milling of the rubout layer.

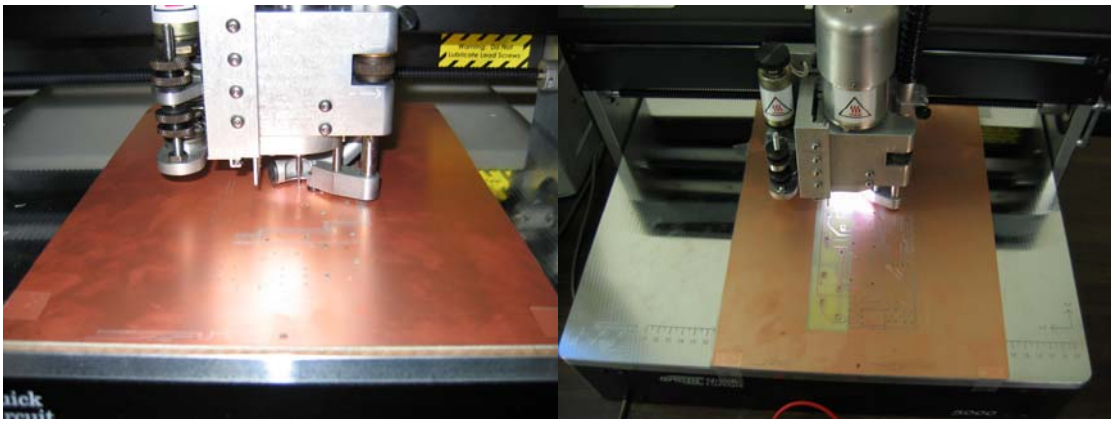

Figure 37: Milling Machine In Use

Once the milling is complete, the board must be tested for shorts and all trace must be verified. Then a rough fit of all components is done to make sure the components fit correctly on the board. Once the board checks out, the component mounting takes place.

7.3 Component Mounting

The component mounting part of the project is very tedious and time consuming. First all, via's have to be soldered. This requires either putting a small piece of solder in the via hole or some other conductive material and soldering the top and bottom of the board. Once that is completed, the surface mount components are soldered to the board. This is done by first applying liquid solder flux to the traces where the component will be placed. Next a line of solder paste is placed to the traces. Then the component is placed in the proper location and the MetCal hot air machine is used. Figure 38 shows the MetCal hot air machine.

Figure 38: MetCal Hot Air Machine

The MetCal machine works by forcing hot air around the surface mount component, adhering the solder to the component and the trace. This machine takes away the stress of having to try and hand solder surface mount components. Once the surface mount components are soldered on, the larger components are then soldered. Last the resistors and capacitors are solderd on. After soldering is complete, a full board test is done looking for shorts or missing connections.

7.4 Fixture Enclosure

The enclosure for the door controller interface system will neatly store the PCB board and allow easy access to the terminal blocks, serial port connection, magnetic card swipe

connection, and JTAG port. The materials that will be used for the prototype enclosure include: ¼" Plexiglas, oak wood trim, mounting screws, and rubber washers. The Plexiglas will be used for the front and back cover. The oak trim will be used for the sides of the enclosure. In future enclosure design, a small aluminum container could work better.

Construction of the door controller interface enclosure will begin with the measuring and cutting of the Plexiglas to required size. Holes will have to be made to both front and back covers so they can be attached to the wooden frame. Next the wooden frame will be constructed. The wooden frame will have to be modified to allow for connection to the wiring blocks and serial port.

CHAPTER EIGHT: TESTING

This chapter discusses the testing involved with the door controller interface system. Both hardware and software testing will be discussed. Test cases will be examined and the testing process will be reviewed.

8.1 Hardware Testing

Hardware testing is a very complex and time consuming process. From the beginning of this project, much time was spent looking for components that would work with each other. The picking of components was done by studying data sheets, verifying proper voltages and amperage requirements and the correct number of inputs and outputs. Even with selection of parts that seem to meet all requirements, problems can arise. Before choosing National Semiconductors DS3658 quad high current peripheral driver, a different line driver was chosen. It was believed this line driver would meet all requirements. When a test was run to energize the relays, the current draw on the line driver was over the specked limit. This error caused a redesign of the PCB board once the current line driver was used.

Once code development started for the MSP430 chip, the use of the developer's kit became very useful. The kit included circuit board with the adapter socket for the microcontroller, JTAG port built in LED and jumper connectors for off board connection was used. The first test run using the MSP430 was turning on and off the on board LED. Once this was accomplished external LED were attached and code was written to turn them off and on. The LED were used to simulate the actual ports turning on and off in the final design. Next code was developed to enable the UART port.

To test communication with the UART the MAX3232 chip had to be introduced and a custom serial port connection had to be made with the MAX3232 chip. Testing was performed by first connecting an oscilloscope probe to the transmit line of the RS232 cable. Next, another oscilloscope probe was attached to the output of the MAX3232. The signals were compared to verify that the signal going in was the signal coming out but at a lower voltage. Once the signal was verified to be correct, a connection was made from the MAX3232 chip to the receive port of the UART on the MSP430. Code was then written to accept commands to turn the onboard LED light off and on. Windows hyper terminal was used to send the commands at 9600 baud rate. Once the board LED was turned off, external LED's were added again and the same test run to simulate the actual outputs. Once communication to the MSP430 was established, inbound and working correctly, communication from the MSP430 to the computer was tested.

Again, a connection was made from the transmit pin of the MSP430 to the MAX3232 chip. An oscilloscope probe was attached to the output to determine if a signal was being produced. Once that signal was confirmed, another oscilloscope probe was attached to the transmit output of the MAX3232 chip. Again, the two signals were compared for correctness. Once the signals were determined to be correct, the connection to the computer was made. Commands were sent from the PC computer using hyper terminal and were echoed back once the MSP430 received the command and sent it back. The two tests for communication proved that everything was working correctly for RS232 communication with the MSP430.

After the communication with the MSP430 was deemed to work correctly, testing moved on to the relays. First, the test to open and close the circuit of the relay was done. This was done by simply applying five volts and ground to the positive and negative terminal of the relay. Next, the DS DS3658 quad high current peripheral driver was wired on a bread board and

connected to the MSP430 along with the relays. Code was executed on the MSP430 to turn on and off each relay in sequence. This worked correctly the first time.

The final bit of testing on the hardware was done when all the components had been soldered to the PCB board. First all, components had to be checked for shorts. Once no shorts were found, then a connectivity test had to be made for each pin of every device to the correct location. This turned up several bad solders and a few mistakes on the PCB itself. The mistakes on the PCB board required jumper wires to be soldered on to correct problems. Once all soldering mistakes were found and corrected, five volts and ground were connected to the board. Next, the board ground was tested to verify everything was working correctly. Also, all parts were felt for extra heat dissipation.

When it appeared all parts were working correctly, a new program was loaded into the MSP430. This also was another test case. There was no way to tell if the onboard JTAG port was working correctly without changing the program. The program loaded, turned on and off the relay outputs. This test worked successfully and proved the JTAG port worked correctly and that all connections between the MSP430 and relays were working correctly.

Overall, the hardware testing should be considered much more complex and time consuming. Without a multi-meter and oscilloscope, debugging hardware problems would be very difficult. Even with the multi-meter and oscilloscope, errors can still occur in code that controls the microcontroller. Hardware testing has been one of the most time consuming tasks of this project.

8.2 Software Testing

The process of software testing can be just as complex as hardware testing, but usually easier to accomplish. Normally, software testing is done first by writing test cases and then executing those cases. In many software organizations, the test cases are run by someone that is familiar with the project, but did not help write the code they are testing. Also, fixing coding defects involves the use of compliers tools, without external tools. In some terms, fixing code problems can be less complex than hardware bugs.

Before testing begins, a test case must be written. The test case should describe the type of environment the test is run in, stopping criteria along with many other options. The stopping criteria used for testing consisted of three points. The first point, testing will only stop upon encountering a fatal error. Fatal errors will be fixed and then retested. If no errors are found during the testing phase, a beta version of the software will be released. The last stopping criteria is, a product is considered delivered when the client can complete common scenarios without any errors in a complete session. Test cases need to be thorough enough to cover all possible cases that could occur with the software. For the door controller interface system test, cases had to be developed for both the client and server. The table lists some of the test cases run on the client and server.

Table 11: Test Cases for PC Software

Test Case #	Test Objective	Test Description	Results
01	Client connects to server	Server is started and the client attempts to connect	Passed
02	Multiple clients connect to server	Once client is connected to the server. A second client attempts to connect	Passed
03	Card reader is being read properly by server	A card is swiped and the ID and the data is read from the card reader and stored in the server database	Passed
04	Client disconnects from the server	The client is able to successfully disconnect from the server	Passed
05	Client is able to send commands to the MSP430	Send command to unlock and lock doors	Passed
06	Server is able to unlock or lock individual doors	Send command to client to unlock or lock a door	Passed
07	Server updates database on schedule change	Change the time in the schedule portion of the sever, verify update to database.	Passed
08	Logging of request to gain access	Verify the database contains the user information for request to gain access	Passed
09	Add and remove users from access list	Add or remove a user from the database via the server GUI	Passed
10	Send new time stamp to client	Send the new unlock and lock time to client when changed	Passed

While the above test cases are just a small sample of things to test between the client server software, they give a perspective to the detail that must be used when testing. With any new upgrades to the software, regressing testing will need to be completed to verify current changes did not affect the working baseline. Always new test cases should be generated to test any new additions to the software.

CHAPTER NINE: CONCLUSION

The development of door controller interface system is the start of a new entry into facility access. The system has the ability to allow facility management to gain control over exterior doors and possibly grow to control all doors within the facility. The system developed provides a working solution to the current problem that the two Engineering buildings are having.

The door controller interface system is able to provide facility management with an easy interface to control all exterior doors. User will be able to unlock and lock exterior doors from server, update times that doors should be locked or unlocked. Also, a record of all users trying to gain access to the facility will be kept. This log will contain the users name, card number, data and time they wanted access, and if access was granted. All of the above listed features give control back to facility management on whom and when people can enter the facility.

The current door interface system is a great start to solving the Colleges facility access problem, but many hardware and software upgrades could be used in the future. The hardware for the exterior doors needs be controlled all by the microcontroller. The currently used laptop should be removed from the equation and all data should be stored on onboard memory on the interface board. Also, a direct Ethernet communication link should be made with the microcontroller. Although these features are not implemented on this version, they are very doable. Another upgrade for the hardware should include the integration for all interior doors. This would be a major upgrade for the interior doors, but a great addition. For the software, if the client applicant was removed the server would need modifications to communicate directly with the microcontroller. Other improvements that could be made to the sever are account

creation. Accounts could possibly be created by just swiping the user's card. If internal doors were added to the system, an access list would need to be created on who should have access to certain doors. This type of system might be based on how Microsoft creates active directors with groups and users. Ultimately, advancements to both the hardware and software are only limited by the user's needs and the technology available.

Facility access, mostly likely, will never go away. Facility management is going to continue to become more and more concerned with who is entering and leaving buildings. Access systems in the future might also control other uses of the building, such as heating and air conditioning, power usage, and computer management. The idea of facility access might one day be the key to facility management.

LIST OF REFERENCES

1. MSP430 Ultra-Low Power MCUs, Texas Instruments publication SLAB034F, 4Q 2003
2. MSP430x1xx Family User's Guide, Texas Instruments publications SLAU046C, 2003.
3. MSP430x11x2, MSP430x12x2 Mixed Signal Microcontroller, Texas Instruments data sheet, SLAS361B, March 2003
4. C. Nagy, Embedded Systems Design using the TI MSP430 Series, Elsevier Science, 2003
5. MAX3232 3-V to 5.5-V Multichannel RS-232 Line Driver/Receiver, Texas Instruments data sheet, SLLS4101, January 2004
6. Interface Circuits for TIA/EIA-232-F Design Notes, Texas Instruments publications SLLA037A, September 2002
7. DS3658 Quad High Current Peripheral Driver, National Semiconductors data sheet, DS005819, 2001
8. Power PCB Relay, Omron data sheet, Catalog Number J03RAD2
9. Port Powered Insertion Reader Technical Reference Manual, Magtek Manual Part Number 99875123 Rev 16, June 2003
10. UAW500S data sheet, Cosel