DESIGN OF POWER MANAGEMENT FOR

PORTABLE COLPOSCOPE

by

CHRIS A. CACERES, B.S.E.E.

A THESIS

IN

ELECTRICAL ENGINEERING

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

Approved

Dr. Brian Nutter
Chairperson of the Committee

Dr. Sunanda Mitra

Accepted

Dr. John Borrelli
Dean of the Graduate School

May, 2005

ACKNOWLEDGEMENTS

First I would like to thank Texas Tech University for the opportunities it has given me. It has been a very challenging experience obtaining my degrees here at Tech, but the time I have spent here in Lubbock meeting new friends while learning has been the best time of my life. I would not trade it for anything.

Next, I would like to thank Hari P. Nayar for his help in explaining the system described in this paper as well as quickly answering any questions that I had pertaining to the system. I would like to thank the technical support teams at Maxim Semiconductor, Microchip, and Inspired Energy. Each of these support teams helped me overcome countless hurdles regarding the devices used in the system.

My thesis advisor, Dr. Brian Nutter, has given me some of my best experiences here at Tech. He has a seemingly limitless amount of knowledge, but he would not answer the many questions I had right away…he would make me think about it and try to come up with the solution myself. This process has been extremely rewarding, because it forced me to think and sometimes come up with answers that I didn't think that I knew. This is truly what an engineer will do to solve problems, so I am very grateful for the help Dr. Nutter has provided me. Dr. Sunanda Mitra has always believed in me, and I want to thank her for supporting me throughout my undergraduate as well as graduate years at Tech, especially when it seemed like I would not be able to finish.

I would like to thank my parents who were always supportive and pushed me to work harder at all times. I would be nowhere without them. I would also like to thank Amber Kaiser who also has always believed in me and encouraged me in dark times. I will never forget that she stood by me and knew I would finish my work, even when the end was nowhere in sight.

ABSTRACT

The work in this paper continues the development of a portable power supply suitable for use with a portable colposcope, an instrument used by doctors to diagnose whether a woman has cervical cancer. This paper explains how the system works, but it is software-related and describes the program that was written for the system. The program is intended to manage the system efficiently, allowing the portable colposcope to function properly for the longest amount of time possible without repair. Among the program's responsibilities are 1) choosing whether the battery packs which allow the device to be portable, grid power, or automobile power is the appropriate power source to run the system, 2) re-charging depleted battery packs when automobile or grid power is available, and 3) taking appropriate action when any of the battery packs notifies the system of a critical condition or alarm situation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

A colposcope is a medical instrument used by a physician to view the human cervix. This allows the physician to see if any abnormalities are apparent and, if so, schedule more tests for the patient. Colposcopes that are found in physicians' offices today are quite bulky and expensive, which is why they are not readily available to under-developed countries. The work in this paper continues the development of a portable unit, which would be less expensive and easy to move to and from different destinations. To be truly portable, the system must include re-chargeable battery packs that can power the system. The battery packs will power the system when necessary but, when available, grid or automobile power will run the system, giving the battery packs the chance to re-charge. The work done previously by Hari P. Nayar is hardware-related, and the work here is software-related, but the framework for understanding the choices that were made, the design of the system, and how it works is laid down in Mr. Nayar's paper. Therefore, the first three chapters of this paper will draw very heavily on Mr. Nayar's work. For any information about the system that may not be explained in sufficient detail in this paper, refer to Mr. Nayar's paper [1].

1.1 Powering the system – Smart Battery Packs

A "Smart Battery" is a battery that is capable of providing the user with vital information, such as the battery's current status. Simple batteries just power a system and do not provide any information such as the capacity left or estimated time until they will be fully discharged. This is a drawback to many systems that could benefit from such information, allowing the system to make more intelligent decisions, thus optimizing system performance and life. This is why Smart Batteries were created and are now implemented in many systems. One notable example is a laptop computer. If the computer is running low on battery capacity, then the Smart Battery can notify the system's host, which will then notify the appropriate hardware to warn the user before

shut down measures are initiated. Once the user is warned, he or she can save his or her work before the system powers itself down, preventing data loss.

These Smart Battery packs are the battery packs that will power the system when it is without the option of grid or automobile power. With the Smart Battery packs, the host controller of this system must make intelligent decisions about which battery packs will charge or power the system. This will allow the battery packs to wear equally, optimizing the lifespan of the system. The host controller will also be able to alert the user when necessary via the information that it receives from the Smart Battery packs.

Choice of Battery Chemistry

Size and weight determine how much energy a battery can hold, which limit the battery's effectiveness in being a portable source of power. There are several battery chemistries that could be considered for the portable application of this system. They include Nickel Cadmium, Nickel Metal Hydride, Lithium ion and the latest Polymer batteries. Noted in Mr. Nayar's paper:

> Li-ion battery packs can store higher quantities of energy for the same mass when compared to other chemistries. However, the inherent flammable and explosive nature of Li prevents this battery technology from being used in high end applications requiring large amounts of power for extended periods of time. Li-ion batteries are available as Smart Battery packs. With high energy density and built-in intelligence, this technology is well-suited for the current application. Li-ions far outweigh the other technologies in terms of higher charge density per unit mass, broader operating temperature range, lighter weight and lack of need for pack maintenance. [1]

1.2 Battery Charger and Battery Conditioning

Because this is a system with multiple Smart Batteries, it will not operate by just attaching the battery packs, because there is no charger architecture available that can charge and condition multiple Smart Battery packs as this system requires. The system was designed to be very flexible, which is why a chemistry-independent charger was chosen. This way, many different battery technologies can be used, due to the different specifications of different batteries. Also, upgrading the system with better battery

technologies in the future will be easy with the adaptable design that is used. Li-ion batteries are extremely strict in their charging outline, and to charge the Li-ion packs in the best manner, which will assure optimal life for the packs, a Smart Battery charger chip is used. The particular charger chip used for this project is the MAX1667 [4] level 2 charger from Maxim Semiconductor. Level 2 means that the charger chip acts as a slave in the system and must honor all charging requests from the Smart Battery given that the request is possible, i.e. the voltage and current requirements can be met. As charging occurs, the voltage and current that the Smart Battery needs will change, and so it communicates with the charger chip, telling the charger what it wants to receive. Communication is made possible via a protocol called System Management Bus.

System Management Bus

As Mr. Nayar puts in his paper:

> SM Bus (System Management Bus) [2] is a two wire serial bus architecture derived by Intel and Duracell from the I^2C [5] architecture, originally proposed by Phillips. SM Bus allows system designers to connect low speed peripherals on a two wire bus that enables easy communication between devices. In the current application, the SM Bus is the only means of communication between devices in the system. The MAX1667 is SM Bus-compliant, and so is the battery pack selected for this application, the NI2020 [3] from Inspired Energy. The NI2020 is a Li-ion Smart Battery pack capable of communicating all battery parameters through the SM Bus. [1]

1.3 Charging Host – The PIC18F452 Microcontroller

The intelligent decisions made in this system will come from a microcontroller, which will constantly monitor the battery packs and charger chips to receive information that will help it make the decision on what to do next. Since SM Bus is the means of communication in this system, the microcontroller chip must have SM Bus capabilities. This is why the PIC18F452 [10] microcontroller from Microchip was selected. It has I^2C ports, which are readily adaptable for SM Bus communication.

1.4 Hybrid Charger

The MAX1667, as a level 2 charger chip, will always honor the requests of the Smart Battery. However, the microcontroller, as host of the system, actually makes the final determination of which battery will get charged at any particular time. Therefore, in this way, the charging module acts as a level 3 charger because "intelligent" decisions are being made as to who gets charged and when. This is why this system is considered a hybrid system (both level 2 and level 3), because a level 2 charger chip is used but level 3 decisions are made by the host or "intelligence" in the system (PIC18F452).

1.5 Integration and System Architecture

The system layout is similar to the method that is used to interface PCI cards to a computer motherboard. There are four NI2020 Smart Batteries in the system, which require four MAX1667 charger chips to charge them. The charger chips are mounted on their own charger boards. The Smart Batteries will connect to the charger boards, but there must be a motherboard to integrate the four charger boards with the microcontroller.

The motherboard provides the interconnections for the four charger boards and the microcontroller. For this prototype design, the microcontroller has its own board, called the PICDEM 2 PLUS DEMO BOARD. It connects with the motherboard via male and female headers. Figure 1, shown below, will give a visual picture of the system.

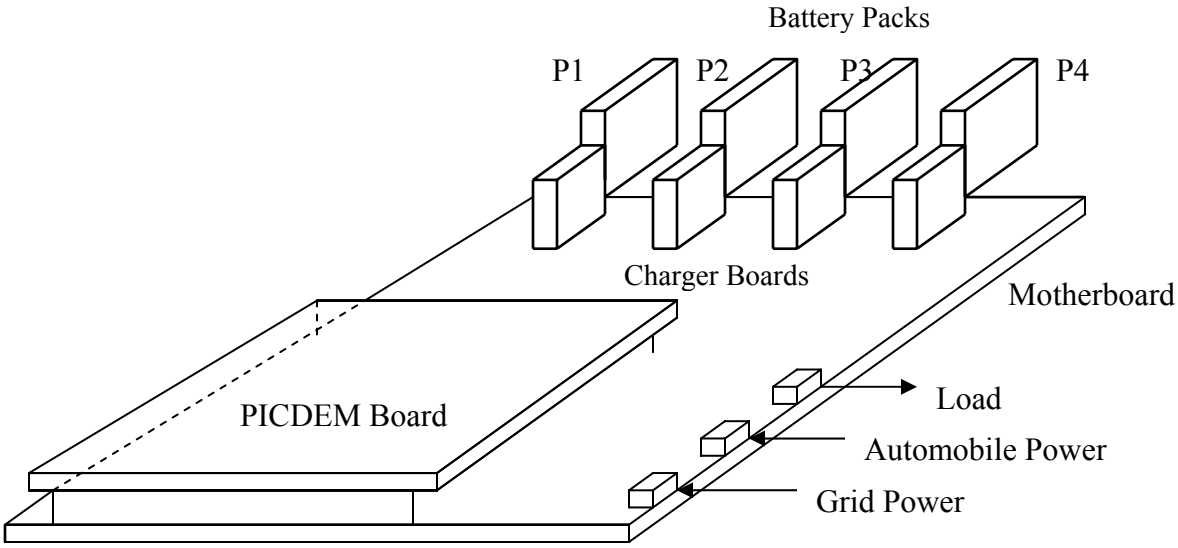

Figure 1. Generic System Diagram [1]

1.6 Additional Applications

The system described in this paper is of course meant for a portable colposcope, but upon removal of the not-yet designed imaging circuitry, this system could be adapted for many different applications. Essentially, the system is a battery charger/discharger, meaning that when necessary, the batteries will power a load or discharge, and can be re-charged with automobile or grid power. So, any portable system with similar power requirements could benefit from this flexible design.

CHAPTER 2
POWER CONSTRAINTS AND BATTERY PACK SOLUTIONS

The portable colposcope described in this paper is intended to work in areas where reliable power sources are not always guaranteed. It was designed with 4 Li-ion battery packs in mind so that the system could power itself for extended intervals without the use of automobile or grid power.

2.1 Power Requirements for Colposcope

As Mr. Nayar states in his paper:

> Commercially available colposcopes operate at very high illumination. An office colposcope offered by Leica operates at 150 Watts (W) of illumination. It is apparent that for the portable unit to operate in the desired market, such high levels of illumination will drastically increase unit cost and lower unit reliability. Indeed, the recharge requirements probably cannot be met within other system constraints. To counter this, the image processing algorithms and illumination strategy will have to compensate for lower light levels and the subsequent increase in shot noise. 40W of illumination is a more realistic power level for the portable device. [1]

There are four batteries in the system, so that the system can provide the 40 W of power for about 7 hours, as will be explained shortly.

2.2 Generic Properties – Li-ion Battery Packs

For the same weight, Li-ions contain more charge than other chemistries, and it is a significant advantage for our portable Smart Battery to hold more charge at lighter weights than other battery chemistries. Also, Li-ion battery packs can perform at a slightly wider temperature range than others although batteries that work in higher than normal temperatures will have some performance issues, such as decline in the charge supplied as well as shortened life of the battery cells. Not only can the Li-ion battery packs work better in more extreme conditions, the negative effects that they will suffer aren't nearly as bad as with other batteries. There is not much that can be done to fix the temperature problems that the unit may face. In fact, in software the only thing that can

be done is to prevent a battery from charging or discharging (powering the system). Fans are not a great idea for this system since we will only call upon the batteries when there are no other sources of power available, and the fans would draw power away from the batteries since the fans themselves need power to operate. Heat sinks will also not work effectively in this system, because there is a very small area on the battery packs where the heat sinks could be placed, and this is not enough area to draw a significant amount of heat away from the packs. Another benefit of the Li-ion Smart Battery is that it has temperature sensors within the unit and can alert the system host when it is too hot. The host simply monitors an alarm condition and will know if the battery has elevated temperatures from this alarm. The host can then take appropriate action either by stopping charge or using another battery to power the system.

The reasons aforementioned are why the Li-ion battery chemistry was chosen and outweigh the fact that this chemistry is more expensive than the other chemistries. The NI2020 [3] Li-ion battery pack is from Inspired Energy. It is a Smart Battery that is appropriate for this application, being SM Bus-compliant, and it can perform in either a level 2 or a level 3 charging scenario. Each battery pack is rated at 10.8 Volts (V) and 6.6 Amp hours (Ah). Since the charger we are using is a level 2 Smart Battery charger, the Smart Battery is responsible for starting communication and for supplying the charging algorithm to the smart charger. Therefore, knowing that the smart charger is an SM Bus Slave device only, it is fairly inexpensive and simple to use.

To supply 40 W at 10.8 V, we require about 3.7 Amps (A), and since the NI2020 is rated at 6.6 Ah, each battery has the ability to power the load for about 1.78 hours, meaning that with four of these battery packs in the system, it is possible to run the system for roughly 7 hours using the battery packs only. The battery packs, however, are not the first choice, because the host microcontroller can determine when automobile or grid power is available and use one of those power sources to power the load as well as charge any batteries that need charging. So from the above calculation, we know that when absolutely necessary, the unit can be portable and operational for up to about 7 hours.

2.3 NI2020 Smart Battery System

The NI2020 Smart Battery is made up of 9 Lithium Ion rechargeable cells of 18650 size, with the cells assembled in a 3 series / 3 parallel (3S 3P) arrangement. Each cell has an average voltage of 3.6 V, over the full discharge period, and a typical capacity of 2.2 Ah, adding up to give the battery pack 10.8 V, on average, and 6.6 Ah. This particular Smart Battery pack follows a constant current, constant voltage (CC-CV) charging pattern. When the battery pack is completely discharged and is requesting charge, it will initially charge at a constant current as the voltage rises. When the battery voltage reaches about 12.6 V, it will then switch over to constant voltage charge, as the current decreases. Once the current gets down to about 220 milliAmps (mA), the battery will be considered fully charged. This charging scheme is shown by the graph in Figure 2.

The bit mapping for the BatteryStatus() register in the NI2020 is also seen below, in Figure 3. This BatteryStatus() register will let the host microcontroller know whether the battery is discharging or not, as well as any critical conditions (alarms) the battery may have. If the battery were fully charged, then the FULLY_CHARGED bit would be set. The FULLY_DISCHARGED bit should never get set in this system, because the software will not allow the battery to power the system if it gets to within 5 minutes of being fully discharged or if the capacity goes down to 5% (0% being fully discharged). The reason for this is to definitely make sure that the battery has charge remaining while we switch to another battery pack to power the system. If a battery is close to being fully discharged and the load requires the full 40 W of power, then it is possible that by the time we notice we need to switch batteries, the battery could get completely depleted of charge, and then the system would go dead because the microcontroller would no longer be powered (the battery would be powering the system, which includes the microcontroller), and we can't switch batteries except under control of the host. This is why we switch batteries while we know the depleted battery has enough charge to power the load.

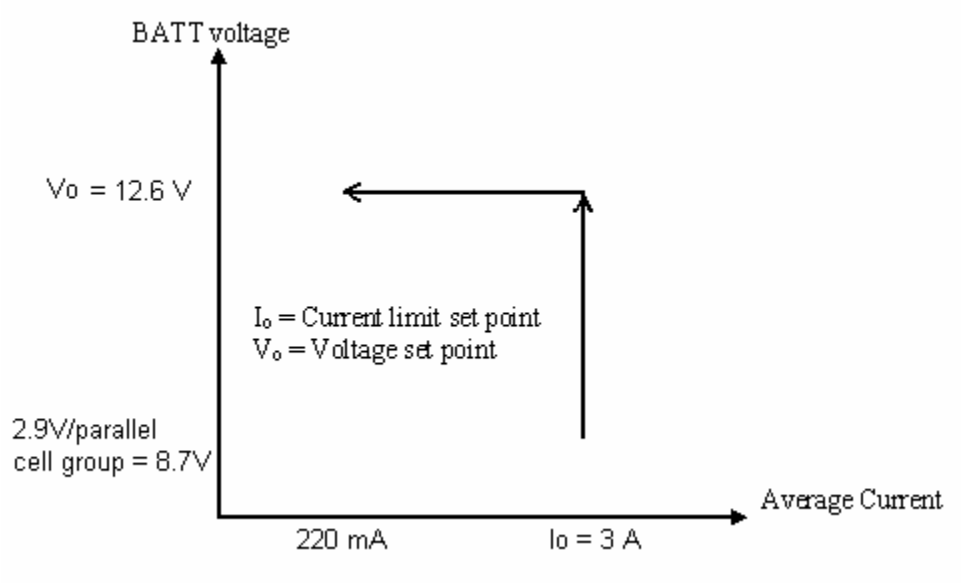

Figure 2. Output V-I characteristics of MAX1667 [1]

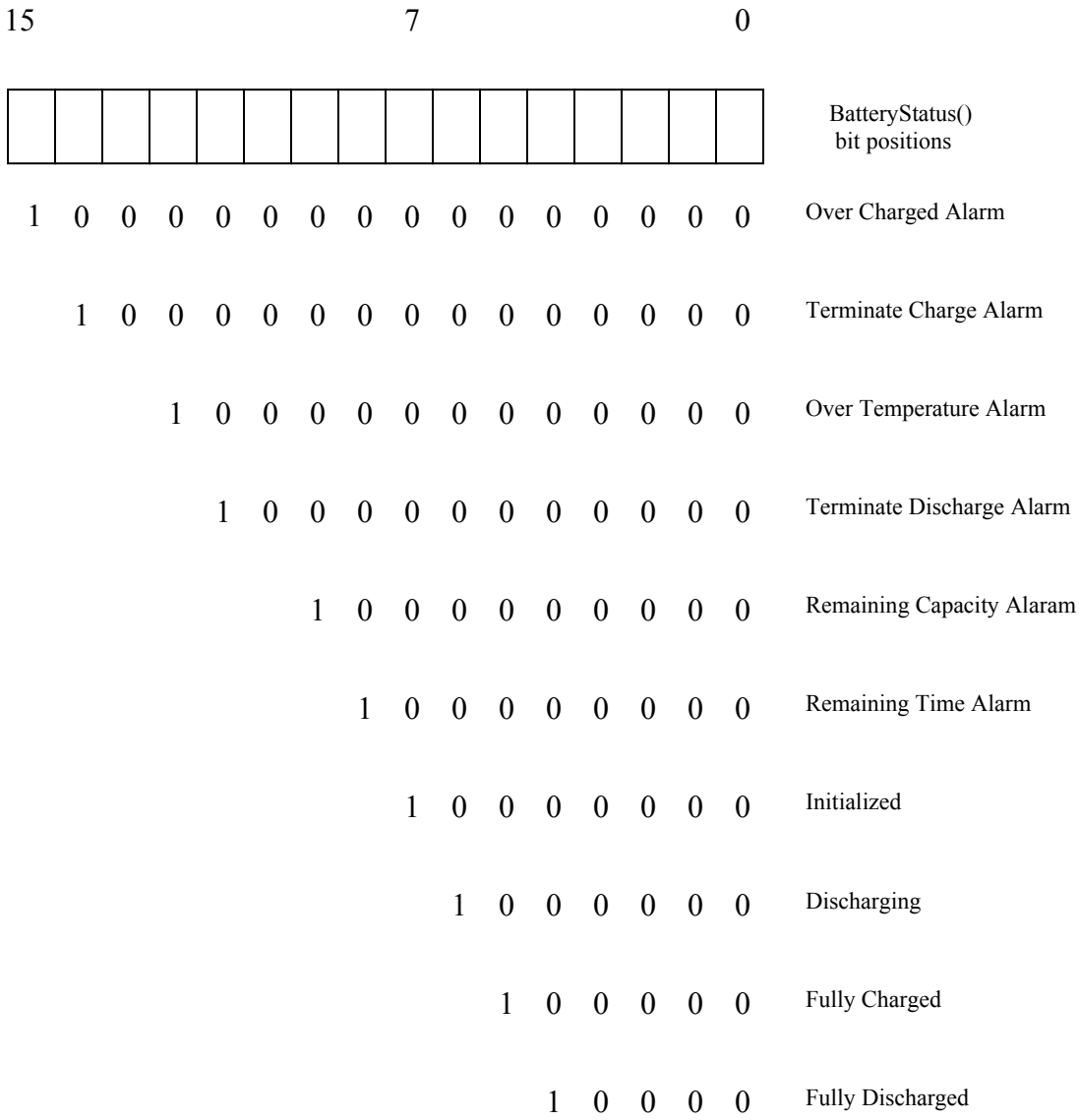

Figure 3. Bit Mapping - Battery Status [1]

The software will provide protection from over-charge and over-discharge, and the battery pack has built-in passive safety devices for protection against short circuit, over-current and over-temperature, as can be seen in Figure 4.

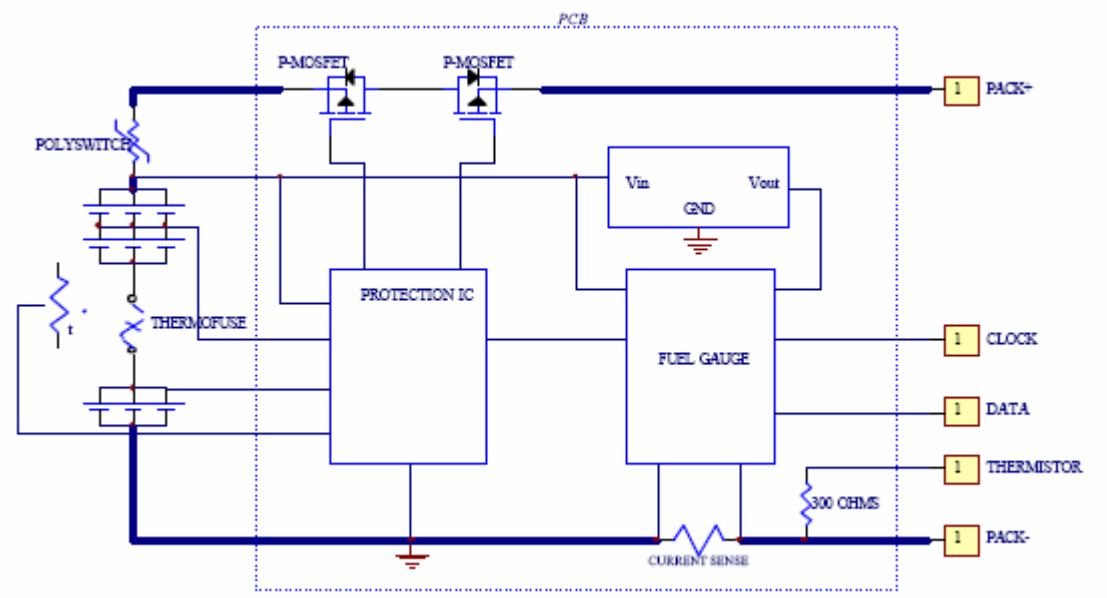

Figure 4. NI2020 Internal Architecture [1]

The battery has three different modes of operation, which are active mode, standby mode and shutdown mode. We will be in active mode if the host microcontroller is communicating with the battery, the battery is powering the system, or the battery is being charged. During this mode, the battery will consume about 1 milliamp of current. When the battery detects no host (no activity on the Bus), and the pack is neither charging nor discharging, then the pack enters Standby mode, consuming no more than 220 microAmps (uA). In our system, this will never happen, because the Smart Battery will consider the Bus active if the clock or data line used to communicate is at 5 V, which it will be as described in Chapter 4. Therefore, the battery will always be in active mode as long as it is not completely depleted of charge. Another problem is that the microcontroller itself can require as much as 25 mA of current while operating. This will drastically reduce the battery's charge much faster than we want. When we know that there is no load to power and auto or grid power is not available, we will make sure that the microcontroller is in sleep mode, which requires much less current. It will jump out of sleep mode every two minutes to monitor the system for any changes and then go back

into sleep mode, assuming there is no load to power and auto or grid power is not available. This way the 25 mA that the microcontroller could possibly require will only be used for a few milliseconds before the microcontroller returns to sleep mode for at least 2 minutes. During sleep mode, the microcontroller will draw no more than a few hundred microamps. The last mode is called Shut-down mode, which will happen if the battery pack's voltage is below 2.4 ± 0.08 V per parallel cell group. The battery would consume less than 1 microamp in this mode, but we should never enter this mode because in software we take careful measures to ensure that the batteries are not depleted too much.

All of the information, including battery parameters, which the host microcontroller needs to know, can be accessed using SM Bus. As noted in Mr. Nayar's paper:

> These parameters provide either factual data or predictive data. Factual data can be measured, such as temperature, pack voltage or charge / discharge current, or it can be a battery characteristic, such as the battery's chemistry. Predictive data is calculated, based on the battery's present state and the battery's characteristics, such as the battery's remaining life at the present rate of drain. Additionally, since the battery has a clock, information can be presented as a rolling average over a fixed interval. [1]

As will be discussed in more detail later, all NI2020 battery packs have the same Slave address on SM Bus, and with four of these battery packs used in the system, it is not feasible to share the same SM Bus communication lines. Therefore, it is not ideal to rely upon one smart charger chip to charge all four Smart Batteries, because if this architecture were chosen, the Bus lines would be extensively toggled to connect the battery pack under charge to the smart charger chip. Also if the system were to only rely on one charger chip, the system would be more prone to error, relying on only one charger chip. These problems are why independent charger boards are designed, with each of these boards being semi-independent level 2 chargers. Each of the 4 charger boards is capable of providing the voltage and current that the batteries may request, given that the host microcontroller permits it. This architecture is called a "4 Charger 4 Pack" or 4C4P configuration, and for our application, the 4C4P configuration is the optimal design.

CHAPTER 3
SYSTEM ARCHITECTURE

The entire system is made up of a motherboard, to which the four charger boards are interfaced, and this chapter explains the complete system architecture as well as why this design is best suited for the portable colposcope.

3.1 Charger Board Architecture

The charger board will manage the charging of the Smart Batteries, and the MAX1667 and associated circuitry necessary for the MAX1667 to perform effectively are mounted on this board. These boards receive their power from the motherboard and are interconnected to the motherboard using devoted headers, which gives each charger board an individual but identical and interchangeable place on the motherboard.

3.1.1 Four-Charger Four-Pack Configuration (4C4P)

As mentioned earlier, it is easier and more reliable to have four of the MAX1667 charger chips, each dedicated to one of the NI2020 Smart Batteries. Each of the Smart batteries has 5 terminals, which are made up of the battery positive terminal (+), the SM Bus clock terminal (SMC), the SM Bus data terminal (SMD), the thermistor terminal (T) and the battery negative terminal(-), and each terminal connects to distinct points on the charger board via a devoted header. Since we have one charger board per battery pack, we can connect each of these terminals to the appropriate pins of the MAX1667 charger chip. This makes the charger board design easier to realize and safer, because high current paths will not be switched. The NI2020 will share the same SM Bus communication lines with the MAX1667, but this will not cause any problems since the Slave addresses of the two devices are different.

Concerning the 4C4P configuration, Mr. Nayar states:

> The biggest advantage of 4C4P is that there is no switching of high current paths. The NI2020 allows a charging current of up to 3A. If one charger were to be used, we would be forced to switch the charge-current path between four different battery packs. This is not an ideal solution, considering safety and board layout issues. On the other hand, when one

> MAX1667 sees one NI2020, there is no switching of high current path. The same charge line can be used for discharge also, because during battery discharge, the MAX1667 is idle. [1]

With the host microcontroller interfaced with the motherboard, the microcontroller has the ability to oversee the four charger chips and four battery packs to regulate charge and discharge. The microcontroller can also sense when alternate sources of power (automobile and grid) are available, so that it can use those sources for powering the load as well as for charging the batteries when necessary. The most important signals connected to the motherboard are SMC, SMD and power for the charger board. SMC and SMD are two of the most important, because these lines are used to communicate with the host, which is why they are connected to the microcontroller. There is a problem, however, since four charger boards are connected to the motherboard, and each of the charger boards requires an SMC and an SMD line. The solution for this is switching connections for the SM Bus lines, which allow the microcontroller to communicate with any of the charger boards it desires using SM Bus.

3.1.2 Charger Board Description

The charger board is considered the main operative unit of the system, because the MAX1667 supplies the power control that is needed to charge Smart Batteries of any chemistry. The MAX1667 will also trigger an interrupt when one of three things happens. These things are 1) the application of power to the charger, 2) the connection or removal of the battery that the charger chip will charge (changing the current state), and 3) the failure to supply the charger chip with enough voltage to properly charge the battery. These will be discussed in more detail in Chapter 5.

The MAX1667 can function when it receives between 7 VDC and 28 VDC. This is perfect for our application, where grid power is presented to the MAX1667 at 15 VDC, and automobile power is presented to the MAX1667 at nominally 15 VDC. The automobile must be on, otherwise the voltage presented to the motherboard will be 12 VDC, and will not be recognized by this system. There are two regulation loops, one for voltage and one for current. The voltage-regulation loop observes the BATT pin of the

MAX1667 to make sure that the voltage on this pin never rises above the voltage set point (V_0). The current-regulation loop will observe the current supplied to the BATT pin to make sure that it never rises above the current-limit set point (I_0). When the MAX1667 device is powered up, V_0 is set at about 18.4 V, and I_0 is set at 7 mA. The NI2020 Smart Battery will alter these values as needed when it is being charged through the SM Bus interface. As mentioned earlier about the Constant-Current Constant-Voltage charging scheme, the current regulation loop will be in control while the voltage is less than V_0. When V_0 is reached, the voltage regulation loop will be in control.

3.1.3 Modifications – Charger Board

For more information on the charger board circuit or any other information on the MAX1667, refer to [1] or [4]. This thesis will, however, describe changes that had to be made to the circuit for it to work properly. Figure 5, seen below, shows a schematic of the charger board circuit suggested in [4]. In the original design, the cathode of diode D4 was supposed to be connected to the cathode of diode D5, but it was not, so a wire was soldered onto the board to establish the connection. Capacitor C3 is rated as a 47 nanoFarad (nF) capacitor, but the component installed was 0.47 nF. The correct capacitors had to be ordered to replace the ones that were already on the charger boards. C6 is actually supposed to be made up of two capacitors in parallel, but only one capacitor was used in the system. To solve this, another capacitor of the same value was placed on top of the first one and connected via wire to establish the parallel connection. The biggest problem, however, was resistor R3, used to sense the thermistor value inside the NI2020 Smart Battery. The application circuit for the MAX1667 has R3 rated at 10 kiloOhms (kΩ), expecting to see a normal thermistor value of about 10 kΩ. The thermistor value inside the NI2020, however, is 300 Ω. Therefore, to match this thermistor value, a 300 Ω resistor was used instead of 10 kΩ for R3. The problem with this, though is that too much current will be drawn out of the REF pin (pin number 9) of the MAX1667, causing it to be unable to charge the battery pack. Therefore, to be able to use the NI2020 Smart Battery, the thermistor terminal of the Smart Battery had to be disconnected so that it would not reach the MAX1667. This was done by standing up

resistor R3 and R4 so that the bottom of the resistors would be only on one pad, meaning the connection from the thermistor terminal does not reach the resistors and thus doesn't reach the MAX1667. Wire was used to make the proper connection from the top of R3 to the top of R4 as well as connect to another 10 kΩ resistor. The other end of this extra resistor goes to the negative terminal (-) of the battery, just as the thermistor does. This 10 kΩ resistor now simulates a 10 kΩ thermistor, so that the MAX1667 thinks that a battery is always connected and always running at an acceptable temperature. This solution was better than the charger boards not working at all, but in the future, circuitry needs to be added to the charger boards that will prevent too much current being drawn out of the REF pin. Because the system was designed around using the NI2020 Smart Batteries, adding this circuitry would be better for this system, rather than choosing a new Smart Battery with a 10 kΩ thermistor. Figure 6 shows a working schematic of the charger board, with all of the corrections made during testing, so that the circuit would work properly.

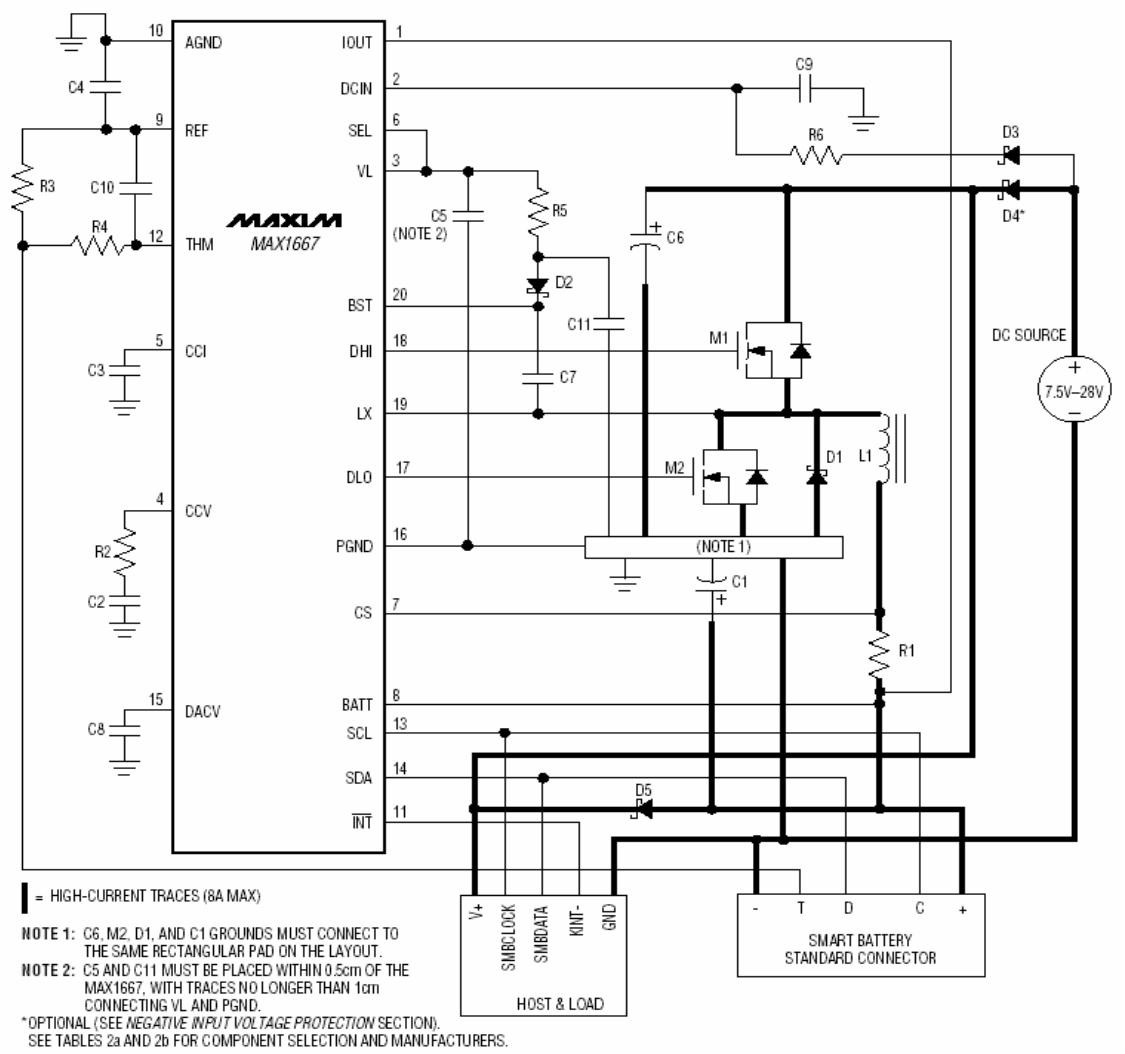

Figure 5. Schematic of Charger Board Circuit [4]

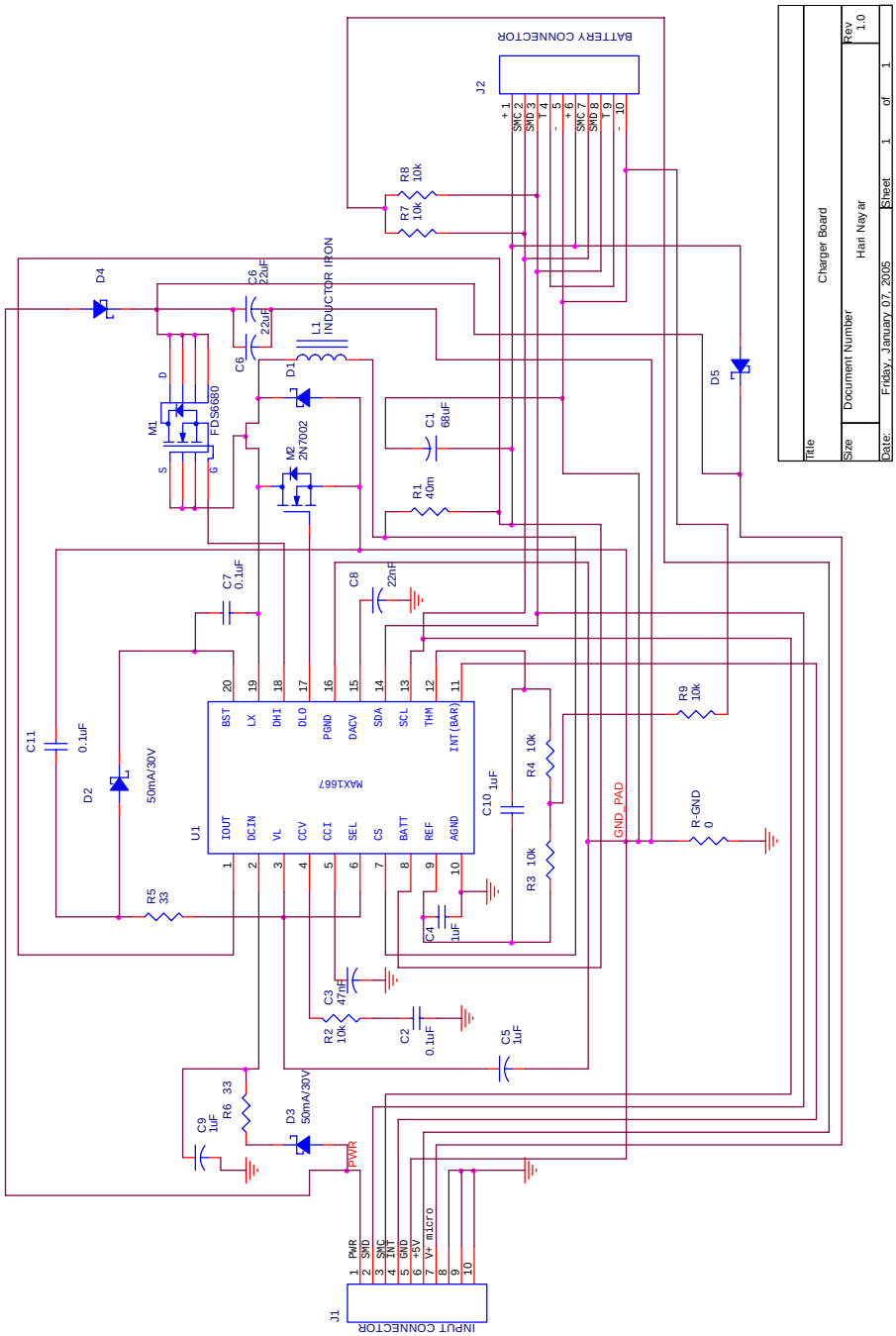

Figure 6. Working Charger Board Schematic

3.2 Motherboard Architecture

The motherboard integrates the system, connecting the microcontroller with the charger boards, and executing tasks that would be difficult to leave to the charger boards.

As mentioned previously, the motherboard provides the interface for the board that houses the PIC18F452 microcontroller, namely the PICDEM 2 PLUS DEMO BOARD. This allows the PIC board to be programmed separate from the system, as it was done, and even used for different applications. This also makes for easy disassembly of each of the different boards for debugging purposes. A block diagram of the motherboard can be seen in Figure 7.

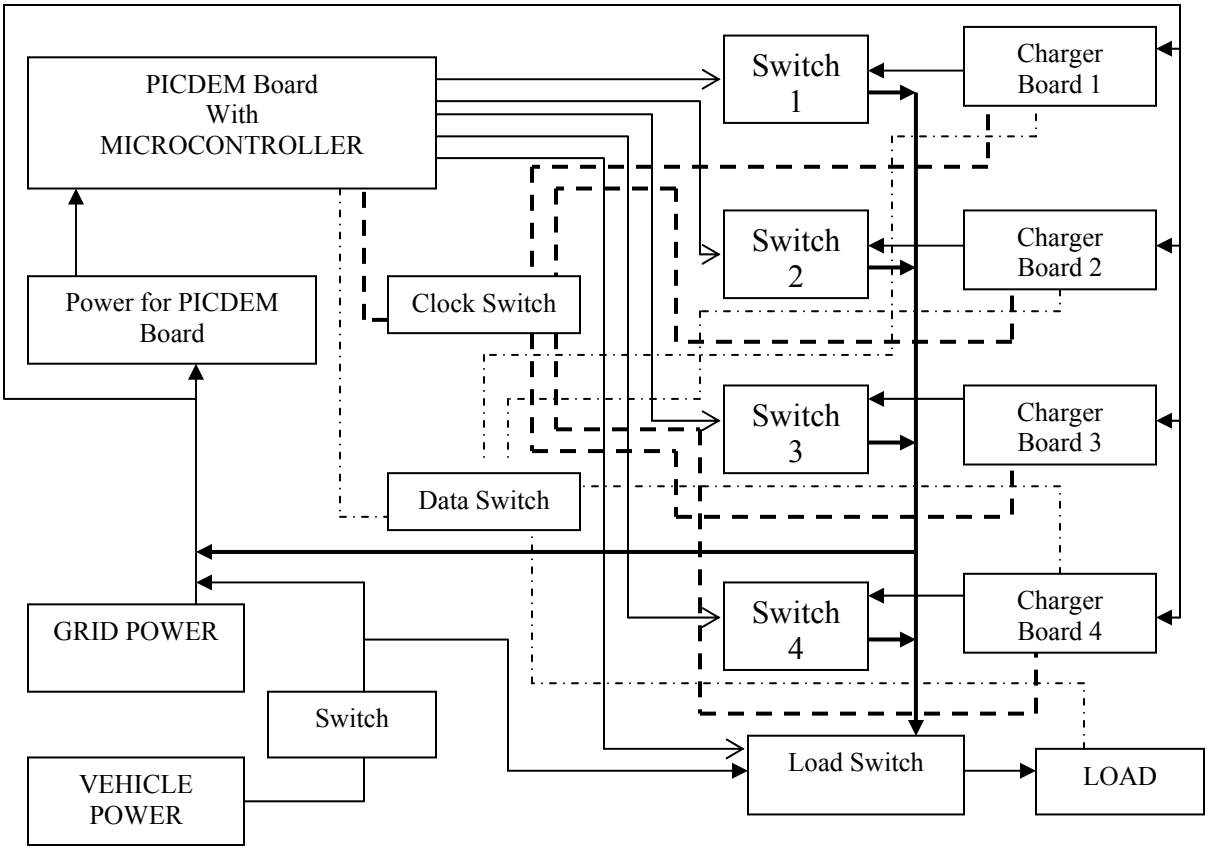

Figure 7. Mother Board Block Diagram [1]

The motherboard performs four main functions, which will be described in the oncoming sections:

- Integrate charger boards to one another and to the host microcontroller
- Integrate the microcontroller with the remainder of the system
- Impart “intelligent” selection of input power for the entire system

- Impart “intelligent” selection of output power for the load (imaging circuitry in our case)

3.2.1 Charger Board Interconnection

As Mr. Nayar says in his paper:

> The mother board acts as the base where all charger boards reside, as shown in Fig 1. All charger boards must be fed with power for charging the battery packs connected to them, and they also must be provided with a discharge path to the load. This is done through the mother board. In addition to charging and discharging paths, the mother board also routes SM Bus signals from the individual charger boards to appropriate pins at the microcontroller. Moreover, each charger board also produces an interrupt line, which originates at the MAX1667. These interrupt lines signal the host when a pack is connected to or disconnected from the charger board. The charger board ground is connected to the primary ground on the mother board, so that the entire system has one common ground. This is achieved at the mother board, where grounds from all the four charger boards and the ground from the PICDEM board are connected to the mother board ground. The battery discharge paths are connected to the load switching mechanism at the mother board. This architecture allows the charger boards to be dedicated chargers and to be exact duplicates of each other. [1]

3.2.2 Microcontroller Interconnection and Bus Switching

The PIC18F452 microcontroller is located on the PICDEM 2 PLUS DEMO BOARD, which was programmed independently of the connection to the motherboard. The PIC was connected any time that code was tested. In the last stage of design for this system, which is beyond the scope of the work done previously as well as the work done in this paper, the microcontroller should be directly integrated with the motherboard as well as the associated circuitry to allow the microcontroller to run as if it were on the DEMO BOARD.

The PIC18F452 microcontroller is accessed through ports on the PICDEM 2 PLUS DEMO BOARD, and these ports provide access to all of the microcontroller’s Input/Output (I/O) pins. The PICDEM board must use the ports to connect to the motherboard. This was made possible by soldering female headers to the motherboard and soldering male headers to the ports on the PICDEM board. This then allows the

PICDEM to connect to the motherboard via the ports. With four Smart Batteries and four smart chargers in the system, there are a total of eight SM Bus lines on the motherboard, but because SM Bus is a two wire serial interface, the microcontroller only needs two of the eight lines. The solution for this is the ADG728 [7] analog switch. This switch can connect any of eight lines to a single line, thus allowing the PIC18F452 microcontroller to talk to one or more of the charger chips and/or Smart Batteries. There are two such switches in the system, one for the four clock lines (clock switch) and one for the four data lines (data switch), each connecting to a single line for the microcontroller.

Because the ADG728 is a serially controlled matrix switch, the switching is done using the SM Bus by writing certain values to the control register, as can be seen in Figure 8. In fact, a one, or high signal (5 V) written to the register will close the switch for the particular bit that was written to in the register. A zero, or low signal (0 V), will open the switch. S1 is connected to header J1-A1, S2 to J1-B1, S3 to J1-C1, and S4 to J1-D1. Therefore, only S1-S4 are used in the register. This will be explained in more detail for the protocol used to talk to the ADG728 in Chapter 4.

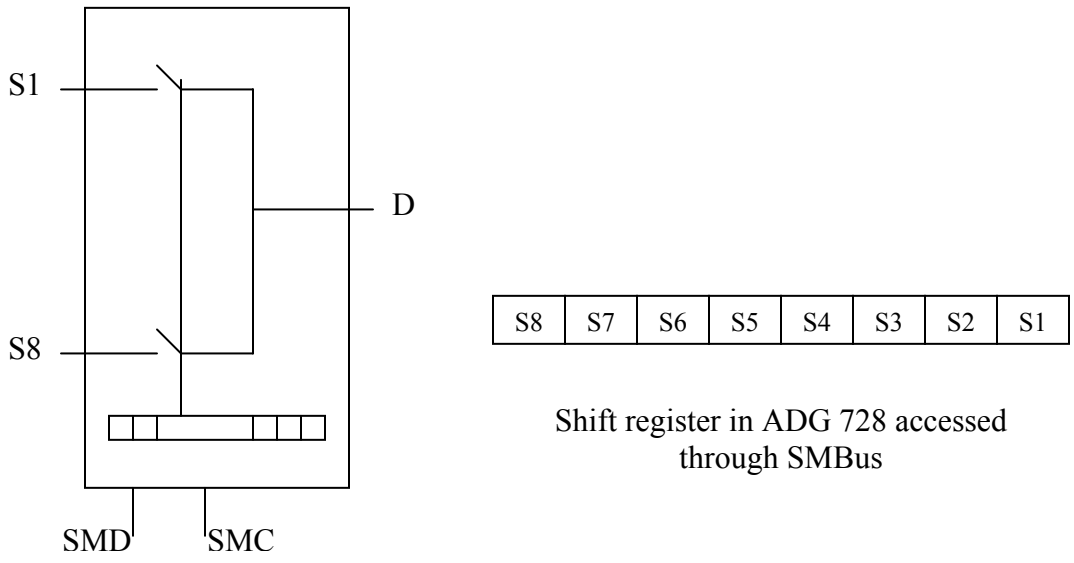

Figure 8. ADG728 Internal Block Diagram and Control Register [1]

3.2.3 Intelligent Power Source Selection and Pack Switching

As mentioned earlier, the portable colposcope will accept power from auto or grid, when available, so that auto or grid will run the system as well as give the system a chance to charge any depleted battery packs. One connection is established on the motherboard for automobile power, and one for grid power. The PICDEM board can be powered by either using a 9-V battery or a connection for a power supply that came with the board. So that the power coming to the PIC will always be reliable, a 9-V output is provided from the motherboard, which can connect to the input connector on the PICDEM board. This 9-V output is accomplished with the LM2940 [13] linear regulator. There are two of these linear regulators on the motherboard, one for the 9 V output going to the PIC, and one for a 5 V output, used to power different devices on the motherboard.

As mentioned earlier, when neither auto nor grid power are available, the battery packs will be used to run the system as well as power the load. As Mr. Nayar notes:

> While powering the load from a pack, a pack's charge state gets depleted quickly in about an hour and half, and the system will have to switch to another pack for continuous functioning. For this reason, the mother board has an active switching mechanism placed in the discharge path of the battery packs. The switching is under direct control of the microcontroller and is implemented using FETs with very low ON resistance. [1]

With all of these power sources available, the motherboard needs to have the ability to choose the source providing the greatest amount of power, especially when the microcontroller is in reset or is powered down. This is accomplished by putting fast switching Schottky diodes in the paths of all of the different power sources, whether it is auto, grid, or any one of the battery packs. The power source that is available, and has the highest voltage, will be the power source selected because this source will effectively turn the diode more ON, which reverse biases the other diodes, not allowing those voltages to power the system. This concept is shown in Figure 9.

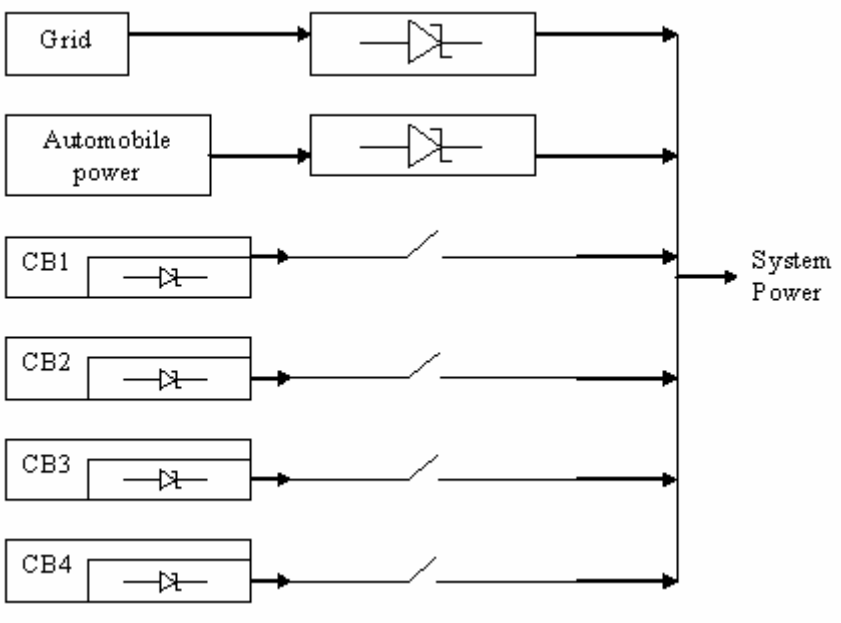

Figure 9. Intelligent Power Source Selection [1]

The Schottky diodes in the paths of grid and automobile power are located on the motherboard. For the battery packs, the Schottky diodes are located on the charger boards, as shown in Figure 8. The FET banks located in the discharge paths of the battery packs form an additional method of switching that can enable or disable a particular battery pack from powering the system. When the system powers up, these FET banks are ON by design. This will be explained in more detail in Chapter 5. For example, if grid power were available (15 VDC), it would be higher than any of the battery packs, which can at maximum be 12.6 VDC. Since grid voltage is higher, it would power the system. If auto or grid were not available, and all of the FET banks in the discharge paths of the batteries were ON, then the battery pack with the highest amount of charger (voltage) would power the system. If multiple battery packs have a similar voltage, however, then they would each provide current to the system. The solution for this problem is explained in Chapter 5 as well.

3.2.4 Intelligent Load Switching

All of the power sources meet together at one point, with only one of them powering the system at any particular time. Next, the power is fed to another FET bank, known as the load switch, as seen in Figure 10 below. On system power up, this FET bank is OFF, so that power cannot reach the load. The microcontroller has to send a signal to turn this FET bank ON so that the load can be powered. The power source that is powering the system will also power the load after the load switch is turned ON. The reason that the system is set up this way is that when auto or grid power is not available, and the microcontroller notices that all of the battery packs have been completely depleted, this design allows the microcontroller to warn the user ahead of time before shutting down the system.

So with this design, the microcontroller has complete control over what happens in the system, which provides us with the most proficient and safe operation of the unit.

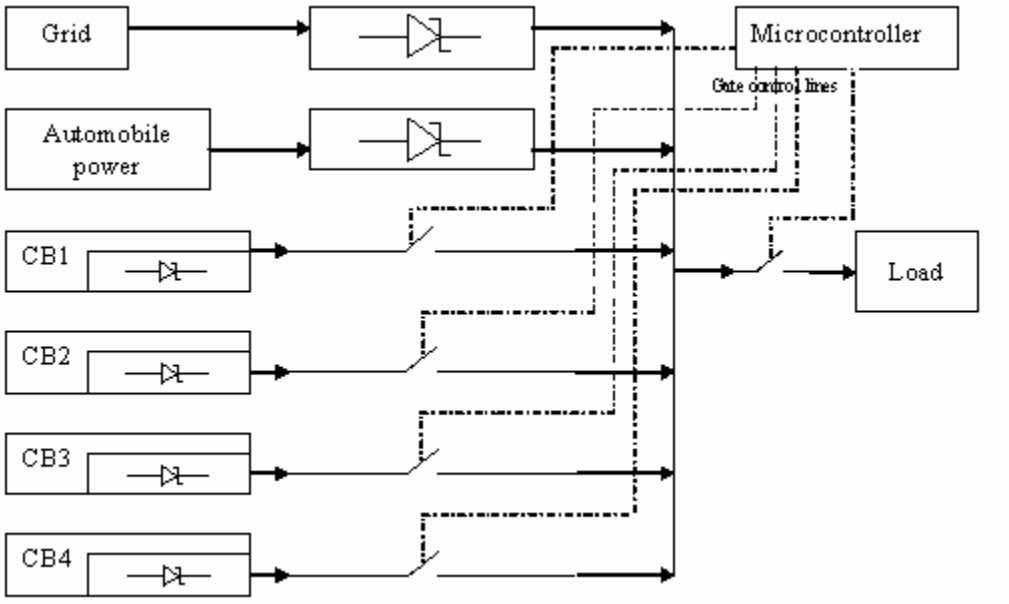

Figure 10. Motherboard Battery Switching Mechanism [1]

3.2.5 Remaining Changes – Motherboard

For a complete schematic description of the motherboard circuit or any other information regarding the hardware design, refer to [1]. There is only one major change

that had to be made to get the circuit to work properly. Initially, the connector for grid power was connected in a way that caused -15 VDC to reach the motherboard. This damaged components so that some things needed to be replaced. To rectify this, a new connector was ordered and connected to the bottom of the board with wire properly so that +15 VDC would power the system. This new connector is used instead of the old one, which still remains on the board, due to difficulty of removing the part. Because there were no changes in the actual design of the motherboard circuit, the schematic found in [1] is the same design used during testing.

There is another problem that needs to be fixed, but was not done so because it did not need to be fixed in order to test the software. The FET bank, known as the load switch, is designed in a way that does not allow full voltage to reach the load. The load will receive about 2 V (typical V_{GS} of the FETs used) less than anticipated. This could possibly prevent the load from receiving the 40 W this system was designed to provide, if powered by the batteries. Therefore, this FET bank needs to be re-designed so that full voltage can reach the load, allowing the load to receive the maximum amount of power available. For more information on the design of the load switch, refer to [1].

CHAPTER 4
SYSTEM MANAGEMENT BUS

System Management Bus, or SM Bus, is the protocol providing the means of communication within this system. The MAX1667 charger chip and NI2020 Smart Battery are both SM Bus-compliant. The ADG728 switches on the motherboard are actually not SM Bus-compliant. They use the I²C protocol from which SM Bus was developed. This is fine, however, since the PIC18F452 microcontroller can switch between using I²C and SM Bus protocols by using the CKE bit in the SSPSTAT register of the microcontroller. The main difference between the two is that the voltage levels for SM Bus are fixed whereas the voltage levels for I²C are scaleable. Also, I²C doesn't use a Command Code, which will be explained in more detail a bit later. For more information on the differences, refer to [2] and [5].

4.1 System Specifications

Any SM Bus device will have a set of commands by which data can be read and written. There are eight possible command protocols, which are Quick Command, Send Byte, Receive Byte, Write Byte/Word, Read Byte/Word, Process, Block Read, and Block Write. The program described in this paper only needs two of the protocols, Read Word and Write Byte/Word. Write Word and Read Word are used with the MAX1667 charger chip and NI2020 Smart Battery. Write Byte is used with the ADG728 switches because it only has one 8-bit register.

The microcontroller acts as the host of the system, monitoring all actions and deciding what is best for the system to maximize its lifespan. Whether it is talking to the switches, a charger chip, or a battery, the microcontroller acts as the Bus Master, with the other devices being Slaves, so that it can read and interpret data to know what the next step will be. The Smart Battery, however, can and will act as Bus Master by requesting voltages and currents from the charger chip when it needs to be charged. The MAX1667 is a level 2 charger, and by design is required to honor all requests, but the host or microcontroller will have the final say in the order of which batteries will be charged,

thereby making the system a level 3 charging-scheme. Optimal system design requires equal wear upon the batteries, to maximize the lifespan for this device as a stand-alone system.

There must be pull-up resistors on the clock and data lines that pull the lines up to 5 V when the Bus is idle. This is a requirement of SM Bus and I²C. Now that this is known, all command protocols begin with a Start bit. This is defined as the data line being pulled low by the Master to 0 V while the clock line remains high at 5 V. For SM Bus, a low signal is considered anything between –0.5 V and 0.6 V. A high signal is anything between 1.4 V and 5.5 V. For this system, low will be pulling the signal to ground or 0 V and high will be releasing the line, which causes the pull-up resistor to pull the signal up to 5 V. Note that the presence of the pull-up resistors affects the battery operating mode.

4.2 Protocol for the ADG728

Figure 11, seen below, shows SM Bus protocol for a Write Byte. Figure 12 shows a timing diagram for an I²C transmission. The SSPIF waveform will be explained in Chapter 5, but the rest of the waveforms aren't necessary because the program was written in C, and functions are used which automatically take care of several steps, including these waveforms, which otherwise would have to be addressed if the program were written in assembly language. The command protocol used for the ADG728 switches will be Write Byte. The first bit sent out is the Start bit, mentioned earlier. Next is the Slave Address. All of the different devices that are SM Bus or I²C compliant will have different Slave Addresses. This is because many systems have multiple devices on the same Bus, and only one pair of devices can communicate. Otherwise the data would be jumbled when two sets of devices try to talk on the same Bus at the same time. When the Master wants to talk to a specific device, it will send out the address pertaining to that device. This way, the device will recognize its address and respond. All devices are "listening" and will only respond when they recognize their own addresses. A problem is posed here, because the system being used has four of the same charger chips and four of the same Smart Batteries. This problem is solved via the ADG728 switches.

Each switch (clock and data) is connected to each of the four charger chips and Smart Batteries. When talking to either a charger chip or Smart Battery, only the line connected to that particular charger chip or Smart Battery will be closed, while the other lines are open, by way of the ADG728s. This way it is only possible that the specific charger chip or Smart Battery that we want to talk to can be reached. Otherwise, any of the four charger chips or Smart Batteries would respond, because they all have the same Slave Address.

Figure 11. SM Bus Write Byte Protocol [2]

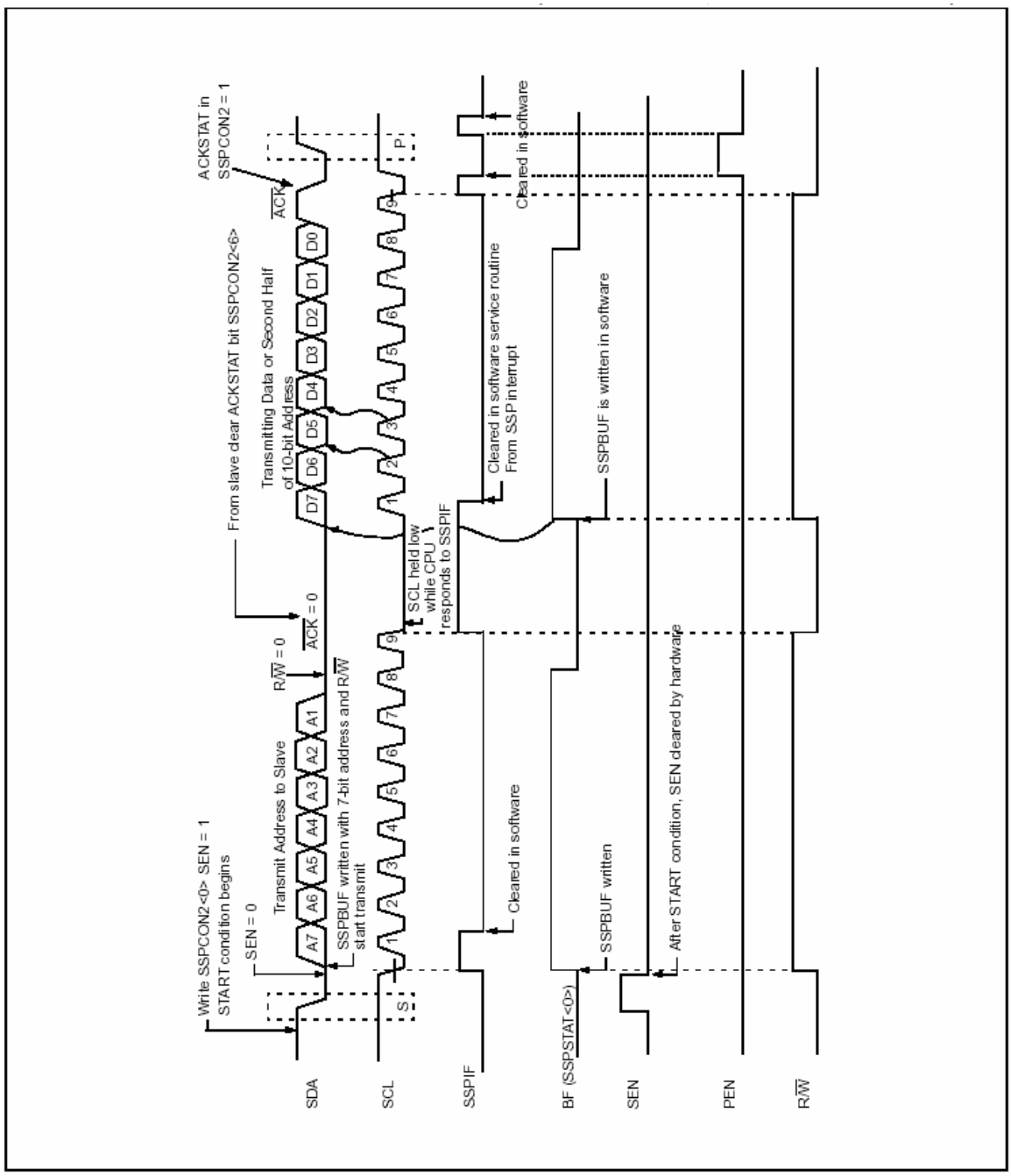

Figure 12. I²C Transmission [10]

When any byte is sent out on the Bus, the Most Significant Bit (MSB) is sent out first. All data bits sent out on the Bus will be clocked in during the high part of the clock pulse. The state of the data line can only change during the low part of the clock pulse. The first 5 bits of the Slave Address for the ADG728 is 0b10011 (0b stands for binary), and the last 2 bits are determined in hardware with the A0 and A1 address pins on the

device. The clock switch, which connects the clock line to the charger chips and Smart Batteries, has its address pins tied high to 5 V, which will represent ones. The data switch, which connects the data line to the charger chips and Smart Batteries, has its address pins tied to ground or 0 V, representing zeroes. The next bit is the Least Significant Bit (LSB) and is called the Read/Write bit. Since we will only be writing to the switches, this bit will always be a Write bit or zero. If we were to read information, that bit would be a one instead. Therefore, this byte of information is made up of the 7 bits of the Slave Address and the Read/Write bit. As a result, when the Master writes to the clock switch, it will send out 0b10011110 or 0x9E (0x stands for Hexadecimal). When the data switch is written, 0b10011000 is used, or 0x98.

After this byte is sent out, the Slave (ADG728) will respond with an Acknowledge (ACK) or Not Acknowledge (NACK). It sends an ACK by pulling the data line low during the ninth clock pulse, after the byte, letting the Master know that the write was successful and that it is ready to continue. If a NACK is sent, the device will leave the line high. This generally happens when the device is busy, and the Master desiring communication with it must try again later. Next, the Master then sends out the Data Byte. This byte is usually reserved for the Command Code, which comes next in the protocol for SM Bus, while I²C doesn't need the Command Code. There are many different types of messages and registers that the host can write to or read from an SM Bus-compliant device. The Command Code describes the type of information the host wants to receive or which register to write to. The ADG728 only has one register to write to or read from, so there is no Command Code necessary for this device. Therefore, the next byte is the Data Byte, which could be 0x00, 0x01, 0x02, 0x04, or 0x08. This is because the register in the ADG728 has 8 bits named S8-S1 from MSB to LSB. The four lines used for each switch are lines S4-S1. Only these five Data Byte values are used, because we only want to have one communication line closed to a particular charger chip or Smart Battery or none closed at all. Because the 4 MSB's of the Data Byte control lines S5-S8 of the ADG728, which are unused, anything could be written to those bits. A zero is chosen for those bits because those lines do not need to be closed if they are not being used. The next bit will be another ACK/NACK, depending on whether the Data

Byte was written successfully or not, and the last bit, which is always the last bit in all Command Protocols, is the Stop bit. Having the Master release the data line, pulling it high, while the clock line is high, sends this bit.

4.3 Protocol for the SM Bus-Compliant Devices

The other two devices that the microcontroller will be communicating with are the MAX1667 charger chip and NI2020 Smart Battery. The Write Word and Read Word Command Protocols are used for these devices. Figure 13 shows the Write Word Protocol. The Write Word begins with a Start bit, followed by the Slave Address and Write bit. In the case of the charger chip, the byte sent will be 0x12 (showing that the Slave Address is 0b0001001). With the Smart Battery, 0x16 is sent (the Slave Address is 0b0001011). Then the charger chip or Smart Battery will send an ACK/NACK. Next is the Command Code. Both of the devices have several Command Codes, which will be discussed later. This is followed by an ACK/NACK, and then comes the data. The Master sends the low Data Byte first, then the high Data Byte. Each Data Byte is followed by an ACK/NACK response by the Slave. Last comes the Stop bit.

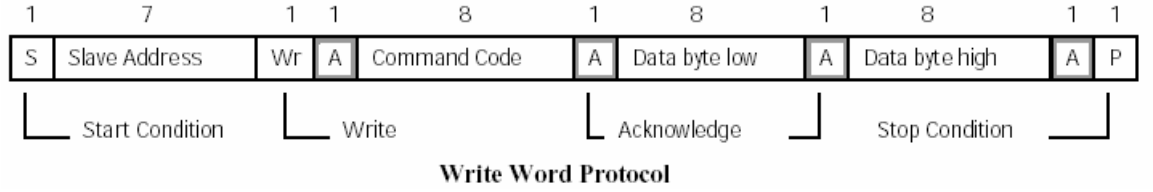

Figure 13. SM Bus Write Word Protocol [2]

Reading information is a little more complicated. Figure 14 shows the Read Word Protocol, and Figure 15 shows the timing diagram for an I²C reception. First comes the Start bit and then the Slave Address followed by the Write bit. For the charger chip, 0x12 will be sent out, and for the Smart Battery, it will be 0x16. Then we have the ACK/NACK followed by the Command Code and then another ACK/NACK. Then a Repeated Start Condition is sent, which is technically a Stop bit followed by a Start bit. Next, the Slave Address is sent again, this time followed by the Read bit. For the charger chip, this byte is 0x13, and for the Smart Battery, 0x17. Then comes the ACK/NACK from the Slave. Assuming that an ACK is sent, the Slave then sends the low Data Byte

out. It is then up to the Master to send the ACK/NACK response, and once the Slave receives an ACK, it sends the high Data Byte. After the high Data Byte is received, the Master should send a NACK, which signifies the end of a read condition. The last bit is as always the Stop bit.

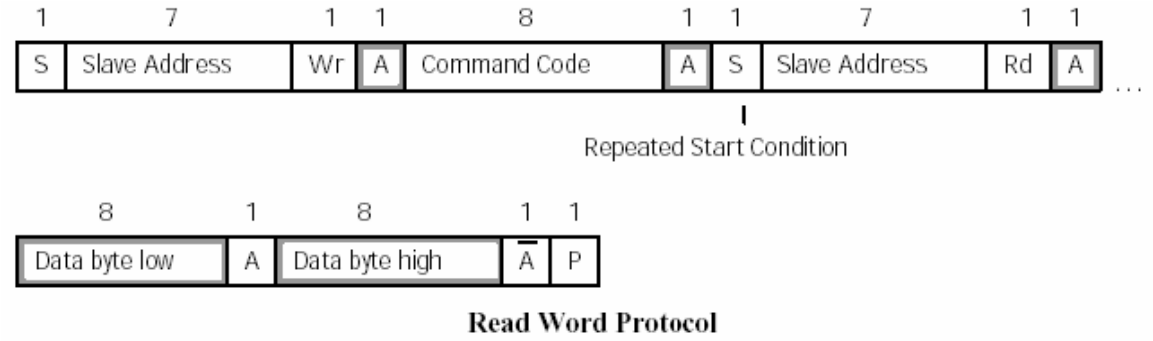

Figure 14. SM Bus Read Word Protocol [2]

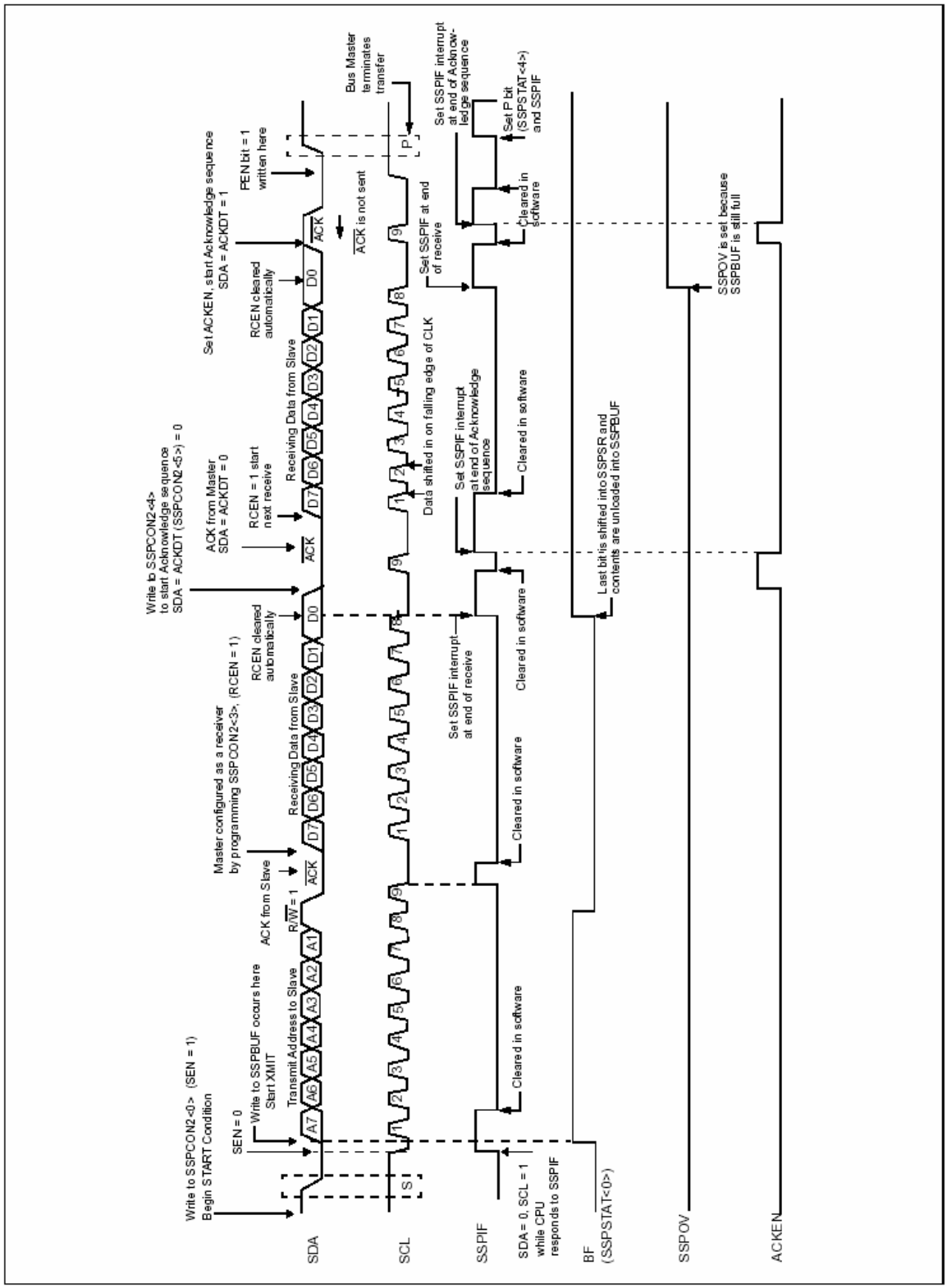

Figure 15. I²C Reception [10]

4.4 Command Codes and I²C Functions

There are many different Command Codes for the Smart Battery. Table 1, seen below, shows most of the Command Codes for the NI2020 Smart Battery. The MAX1667 only has a few of them. For the Write Word format they are ChargerMode() = 0x12, ChargingCurrent() = 0x14, ChargingVoltage() = 0x15, and AlarmWarning() = 0x16. For the Read Word format, there is only ChargerStatus() = 0x13. Some of these will be discussed in more detail in the following chapter. For more information regarding the NI2020 Smart Battery, refer to [3], and refer to [4] for the MAX1667 charger chip.

Host-to-Battery Messages

Function	Command Code	Description	Unit	Access	Default (POR)
ManufacturerAccess()	0x00			r/w	
RemainingCapacityAlarm()	0x01	Remaining Capacity Alarm Threshold .	mAh	r/w	660
RemainingTimeAlarm()	0x02	Remaining Time Alarm Threshold.	minutes	r/w	10
BatteryMode()	0x03	Battery Operational Modes.	Bit flags	r/w	0x0080
AtRate()	0x04	This function is the first half of a two-function call-set used to set the AtRate value used in calculations made by the AtRateTimeToFull(), AtRateTimeToEmpty(), and AtRateOK() functions.	mA	r/w	0
AtRateTimeToFull()	0x05	Returns the predicted remaining time to fully charge the battery at the AtRate() value.	minutes	r	65535
AtRateTimeToEmpty()	0x06	Returns the predicted remaining operating time if the battery is discharged at the AtRate() value.	minutes	r	65535
AtRateOK()	0x07	Returns a Boolean value that indicates whether or not the battery can deliver the AtRate value of additional energy for 10 seconds. If the AtRate() value is zero or positive, the AtRateOK() function will ALWAYS return TRUE.	boolean	r	1
Temperature()	0x08	Returns the pack's internal temperature.	0.1°K	r	
Voltage()	0x09	Returns the battery's voltage (measured at the cell stack)	mV	r	
Current()	0x0a	Returns the current being supplied (or accepted) through the battery's terminals.	mA	r	0
AverageCurrent()	0x0b	Returns a rolling average based upon the last 64 samples of current.	mA	r	0
MaxError()	0x0c	Returns the expected margin of error.	percent	r	10
RelativeStateOfCharge()	0x0d	Returns the predicted remaining battery capacity expressed as a percentage of FullChargeCapacity().	percent	r	0
AbsoluteStateOfCharge()	0x0e	Returns the predicted remaining battery capacity expressed as a percentage of DesignCapacity().	percent	r	0
RemainingCapacity()	0x0f	Returns the predicted remaining battery capacity.	mAh	r	0
FullChargeCapacity()	0x10	Returns the predicted battery capacity when fully charged.	mAh	r	5940
RunTimeToEmpty()	0x11	Returns the predicted remaining battery life at the present rate of discharge.	minutes	r	65535
AverageTimeToEmpty()	0x12	Returns the rolling average of the predicted remaining battery life.	minutes	r	65535
AverageTimeToFull()	0x13	Returns the rolling average of the predicted remaining time until the battery reaches full charge.	minutes	r	65535
ChargingCurrent()	0x14	Returns the battery's desired charging rate.	mA	r	3000
ChargingVoltage()	0x15	Returns the battery's desired charging voltage.	mV	r	12600
BatteryStatus()	0x16	Returns the battery's status word.	Bit flags	r	0x2C0
CycleCount()	0x17	Returns the number of charge/discharge cycles the battery has experienced. A charge/discharge cycle is defined as "starting from a base value equivalent to the battery's highest SOC reached after the battery is no longer accepting current before the present charge/discharge cycle is complete and ending when the battery starts accepting current and its SOC has decreased by 15% or more from that base value."	cycles	r	0
DesignCapacity()	0x18	Returns the theoretical capacity of the new battery.	mAh	r	6600
DesignVoltage()	0x19	Returns the theoretical voltage of a new battery.	mV	r	10800
SpecificationInfo()	0x1a	Returns the version number of the SBDS the battery pack supports, as well as voltage and current scaling information.	Formatted word	r	0x0010
ManufacturerDate()	0x1b	Returns the date the electronics was manufactured.	Formatted word	r	
SerialNumber()	0x1c	Returns the electronics serial number.	number	r	
Reserved	0x1d - 0x1f			r	

Table 1. Command Codes for the NI2020 Smart Battery [3]

The PIC18F452 microcontroller provides all of the I²C functions necessary to communicate with all I²C and SM Bus-compliant devices. The functions provided include, but are not limited to: StartI2C, StopI2C, RestartI2C, ReadI2C, and WriteI2C. The first three do what would be expected, namely put that condition out on the Bus.

ReadI2C will read a byte off the bus from a device with which the microcontroller is communicating. WriteI2C will write out a byte on the bus. This function is used any time a Slave address with Read/Write bit, Command Code, or Data Byte must be sent out. There are many more functions available, some of which will be discussed in the next chapter. For more information, refer to [6].

CHAPTER 5
PROGRAM ARCHITECTURE

The program that was created to manage the system can be broken up into three parts. The first part is called "Auto or Grid." We will be in this part of the program if automobile or grid is available to power the system. The second part is called "Charge." This part of the program takes place when a battery is being charged. This section is technically under "Auto or Grid" because automobile or grid power is necessary to charge the batteries, but this thesis covers these as two separate sections to make things easier to follow when reviewing the flowchart. The third and last section is entitled "Discharge," which is named because a battery will be discharging in this section. If auto or grid power is not detected, it is then logical to assume that a battery pack is powering the system, meaning that one particular battery is discharging.

5.1 Auto or Grid

"Auto or Grid" is needed to bootstrap the system or recover from complete discharge. Therefore, the "Auto or Grid" section will be in the main function of the program. Figure 16 shows the flowchart for the "Auto or Grid" portion of the program. Once the PIC18F452 microcontroller is powered up, the first action taken is to determine the power source. This is done through the microcontroller's Analog to Digital Converter or ADC. The ADC requires 0 to 5 V input, so a voltage divider steps the voltage coming in from auto or grid down to about a fourth of the original voltage. The maximum the auto or grid voltage will be is 15 V, so that the maximum voltage that the ADC will see is about 3.72 V. The ADC has 10-bit precision, meaning that a 0 will represent 0 V and 1023 will represent 5 V. Now, due to the Schottky diodes in the paths of the battery packs, automobile, and grid sources, the highest voltage will be the one powering the system. The battery packs are rated at 12.6 V, when fully charged, with ±1% accuracy. Therefore, the maximum amount that will be seen at the terminals of the battery packs is 12.726 V, which is why 12.75 V is chosen as the level when determining if auto or grid is powering the system. 12.75 V is reduced to 3.164 V after the voltage divider. When

polling the ADC, if it is found that the voltage coming in from auto or grid is higher than 3.164 V, then we decide that auto or grid is powering the system. Otherwise, it must be a battery pack powering the system and we go to the "Discharge" portion of the program.

Auto or Grid

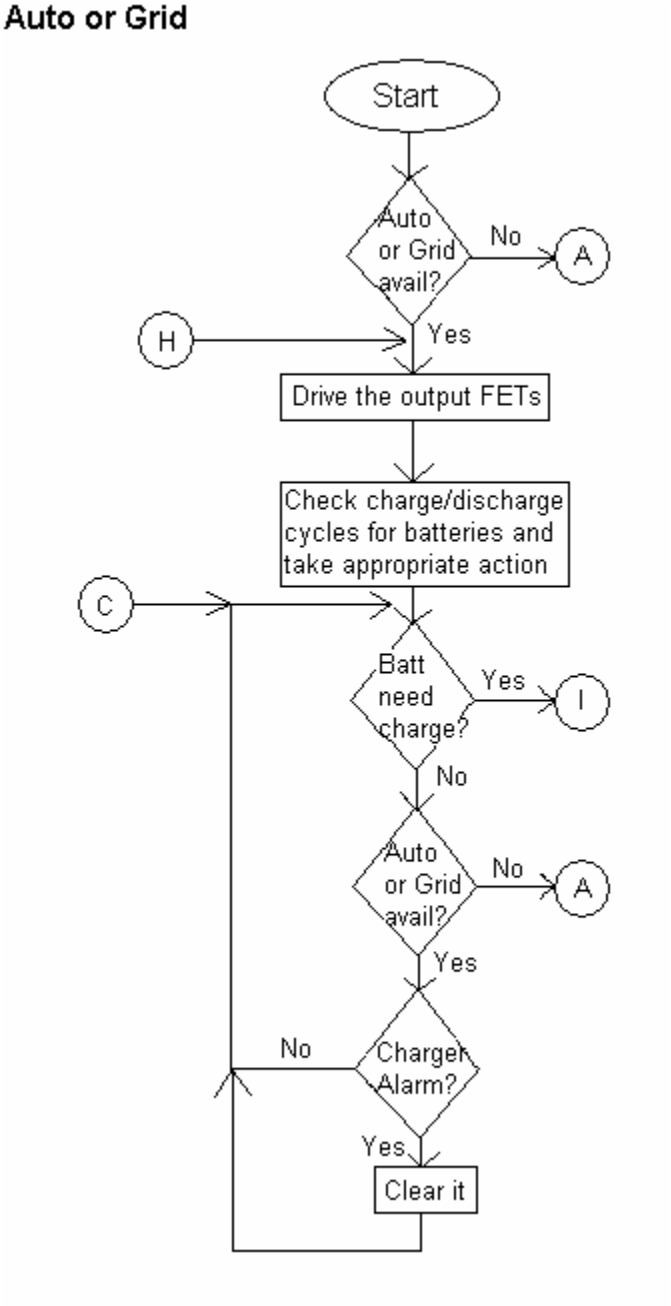

Figure 16. "Auto or Grid" Flowchart

The next step taken under "Auto or Grid", or the main function, is to drive the output Field Effect Transistors, or FETs. On power up of the system, the load does not

receive current because the output FETs are OFF. Therefore, a signal from the microcontroller is necessary to power the load. This is done by simply putting a low signal out on pin RB3, because this is the line that is connected to the gate of the first output FET. For more information on how the output FETs work, refer to [1].

There are also FETs in the discharge paths of the four battery packs. On system power up, these FETs are ON, allowing any one of the four battery packs to power the system. However, with the Schottky diodes in the discharge paths of the battery packs, as well as in the paths of automobile and grid power, even with the FETs ON, if automobile or grid power is available, none of the battery packs will be able to power the load. The next step taken in this part of the program is to check the number of charge/discharge cycles of the four battery packs. This is how we get an estimate as to how much use some packs have received versus others. Since we want equal wear upon the four batteries, in this step of the program, if one of the four battery packs has a higher charge/discharge cycle count than any one of the others, then we turn the FETs OFF for that associated battery pack so that when auto or grid is no longer available, this battery pack will not be able to power the system first. This is done by simply sending a low signal out on pin RD1, RD2, RD3, or RD4, depending on which battery has the higher cycle count. These pins are connected to the gates of one of the FETs in the discharge paths of the battery packs. For more information on how these FETs work, refer to [1].

The next step is to see if a battery needs to be charged or not. There is a bit called INHIBIT_CHARGE in the ChargerMode() register of the MAX1667. If this bit is set, then the charger chip cannot charge the Smart Battery. If the bit is cleared, then the charger chip can. Any time we communicate with either the charger chip or the Smart Battery, this bit will be set, unless we are in the 'Charge' portion of the program and charging is desired or we want to find out which of the batteries is requesting charge. The way we see if a battery must be charged or not is by clearing this bit and then polling the current in the Smart Battery. If the current is positive, it is accepting charge, and we know that this battery must be charged. If the current is zero, the battery is "waiting," and we can assume that it is fully charged. The current will not be negative (meaning it is discharging) at this point because auto or grid will be powering the system, not

allowing any of the battery packs to do so. Therefore, if a battery needs to be charged, we will then go into the "Charge" portion of the program, otherwise we remain under "Auto or Grid." Here, better software control could be designed, based on checking the RelativeStateOfCharge() register in the Smart Battery, to determine if a battery must be charged.

Assuming that no battery needs to be charged, the next step taken is to poll the ADC to make sure that automobile or grid power is still available. If not, we go to "Discharge." If so, we remain in the main function and continue on. We then see if an interrupt has been triggered by any of the MAX1667 charger chips. This interrupt is treated as an alarm, in that when the interrupt is thrown, the program does not go to an interrupt service routine. Instead, a bit in the microcontroller is polled, so from now on, this interrupt will be referred to as the charger alarm. There are three ways the charger alarm can be triggered. The first is if power is applied to DCIN. The second can be triggered when the BATTERY_PRESENT bit in the status register of the charger chip changes. The third is if the POWER_FAIL bit in the status register of the charger chip changes. The way this program was written, the first two ways do not concern us. The third one concerns us because if the POWER_FAIL bit is set, then the MAX1667 charger chip is receiving (DCIN pin) less than 89% of the Smart Battery voltage, which is not enough to charge the battery properly. While this bit is set, we cannot charge a battery. Therefore, if it is known that this bit is set, we will never enter the "Charge" portion of the program. We determine if the charger alarm has been triggered by polling the INT1IF bit in the INTCON3 register of the microcontroller. If it is set, then the charger alarm has occurred. If it is clear, then the charger alarm has not been triggered. Normally, the line tied to the pin involved with this charger alarm (RB1) is high because it has a pull-up resistor. It will go low when the MAX1667 triggers the charger alarm, thus setting the INT1IF bit. The charger alarm is cleared by clearing the INT1IF bit in software as well as issuing a ChargerStatus() command to the MAX1667, so that it will release the line to be pulled high again. After the charger alarm portion of "Auto or Grid," we then loop back to see if a battery needs to be charged. We will remain in this

loop until either a battery needs to be charged or automobile or grid power is no longer available.

One might think that it would be necessary to check the battery for alarms that have been thrown, but checking the battery for alarms is not necessary in this part of the program. The reason is because if a battery throws an alarm, the only action that can be taken from the microcontroller as host of the system is to make sure that that particular battery is not being charged or is not discharging (powering the system). Because we are under the "Auto or Grid" section of the program, no battery is being charged, and because automobile or grid is powering the system, no battery will be discharging. Therefore, it is not necessary to check for alarms in the main function of our program.

5.2 Charge

We will be in this part of the program if it has been determined that at least one of the batteries is requesting charge. Figure 17, seen below, shows the flowchart for this section of the program. The first step taken in this part of the program is to determine which battery to charge and to allow that one to charge. Recall that there are only two steps that must be taken to charge a battery. First, one needs to make sure that the INHIBIT_CHARGE bit in the ChargerMode() register of the MAX1667 is clear. Second, one needs to close communication lines (clock and data) for the particular battery to be charged. This allows that Smart Battery to broadcast the voltage and current that it wants to receive to start charging. The Smart Battery will broadcast this information every 15 seconds to make sure the charger chip knows what it wants. After charging has started, the only way that the microcontroller can stop charge is by setting the INHIBIT_CHARGE bit. The way we determine which battery to charge is by allowing each one to charge and then seeing if it starts charging. If the battery does start charging we get a cycle count from that battery to see how many charge/discharge cycles it has gone through. If more than one battery is requesting charge, the one we pick to charge is the one with the fewest charge/discharge cycles. If two or more batteries have the same number of cycles, the preference is the battery connected to header J1-A1 (or battery A1), then B1, then C1, and then D1.

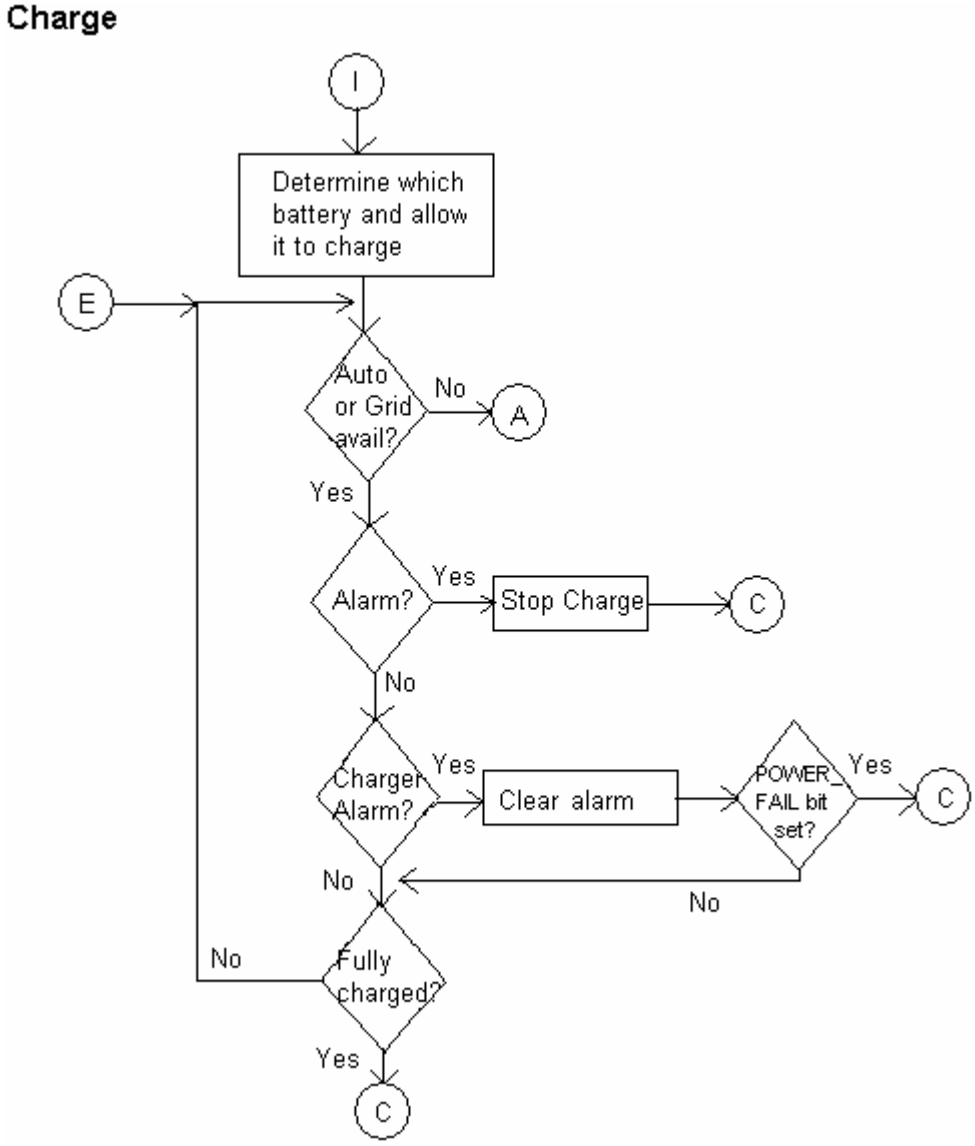

Figure 17. "Charge" Flowchart

The next step in the "Charge" portion of the program is to make sure that automobile or grid power is still available. If not, we go to the "Discharge" section of the program, but if so we continue on. The next step is to check to see if the battery has thrown an alarm or if the charger alarm has occurred. We discussed the charger alarm earlier and so the same procedure is taken if the charger alarm occurs. There are six different types of alarms that the battery can let us know about. The way we check them is to look for certain bits in the status register in the Smart Battery.

The OVER_CHARGED_ALARM, which is bit 15, lets us know that the battery is being charged beyond an end-of-charge indication. The way we fix this problem is by inhibiting charge for that battery. This alarm should never happen since we constantly monitor the battery when charging, and when it is fully charged, we inhibit charge. If this alarm were to happen, we would return to the "Auto or Grid" section. Bit 14 represents the TERMINATE_CHARGE_ALARM. This bit is set when the battery is charging and it notices that either the voltage or current is too high. We fix this problem by stopping charge as well. We should never see this alarm, but if so, we will return to the "Auto or Grid" section. Bit 12 is the OVER_TEMP_ALARM. This bit is set when the battery temperature is above operating range, which is above 54°C while charging, or above 70°C while discharging. It is cleared when the temperature drops below 46°C while charging, or below 65°C while discharging. There are only two ways the microcontroller can take action when this bit is set. If the battery is powering the system, it can switch to another battery, or if the battery is being charged, it can simply stop charging the battery. In other words, if under "Discharge," we will remain in this section and use another battery, or if under "Charge," we will stop charge and return to the "Auto or Grid" section, where we will determine if another battery can be charged. The next alarm is the TERMINATE_DISCHARGE_ALARM, or bit 11. This alarm happens when the battery has supplied all of the charge it can without being damaged. This is solved by using another battery to power the system. This alarm should never happen because as will be mentioned shortly, we will stop powering the system with a certain battery once it is within 5 minutes of being fully discharged or it is at 5% of its rated capacity. If this alarm were to happen, however, we would remain in the "Discharge" section and use another battery to power the system. Bit 9 is the REMAINING_CAPACITY_ALARM, and Bit 8 is the REMAINING_TIME_ALARM. These bits will be set when the capacity reaches 10% or the time reaches 10 minutes until fully discharged. These alarms can be changed, however, and to get more power and time out of each battery, the capacity alarm is set to 5%, and the time alarm is set to 5 minutes. In our case, the capacity alarm will clear when the battery capacity rises above 5%, and the time alarm will clear when the remaining time until being fully discharged goes above 5 minutes. The action that is

taken by the microcontroller when one of these alarms is seen is to stop powering the system with that battery and use another. The alarms for that battery might remain for a while in this system because they cannot be cleared until the battery receives charge. If these alarms are seen, we would remain in the "Discharge" section.

If an alarm is thrown while we are under the "Charge" section, it is going to cause us to stop charging the battery and head back to the "Auto or Grid" section. The last step in the "Charge" portion of the program is to see if the battery is fully charged. This is done by checking the FULLY_CHARGED bit in the status register of the Smart Battery. This would be bit 5 in that register. If it is set, then it is fully charged, if clear, then it is not fully charged. If we determine that a battery is fully charged, then we will return to the "Auto or Grid" section and see if another battery can be charged and take the appropriate action. If the battery that we are charging is not fully charged, we loop back in the "Charge" section to see if automobile or grid power is still available and continue on. We will remain in this loop until either automobile or grid power is no longer available, an alarm or interrupt causes us to exit this section, or if we determine that the battery we are charging is fully charged.

5.3 Discharge

Figure 18 shows the flowchart for the "Discharge" section of the program. The first step taken in this section is to drive the output FETs. This is done, as mentioned earlier, by sending a low signal out on pin RB3, because this is the line that is connected to the gate of the first output FET. The reason for this step is that when the system is first powered up, the power source may be the batteries, in which case we have not driven the output FETs to power the load yet. Therefore, we need to do this as a precaution. The next step is to make sure that no battery is being charged. It is possible to go into the 'Discharge' section from the 'Charge' section if automobile or grid power is suddenly no longer available. We do not want a battery to charge another battery, because this is a waste of the battery power and will cause the battery powering the system to deplete very rapidly since it will be powering the load as well as providing the charge necessary for the charging battery.

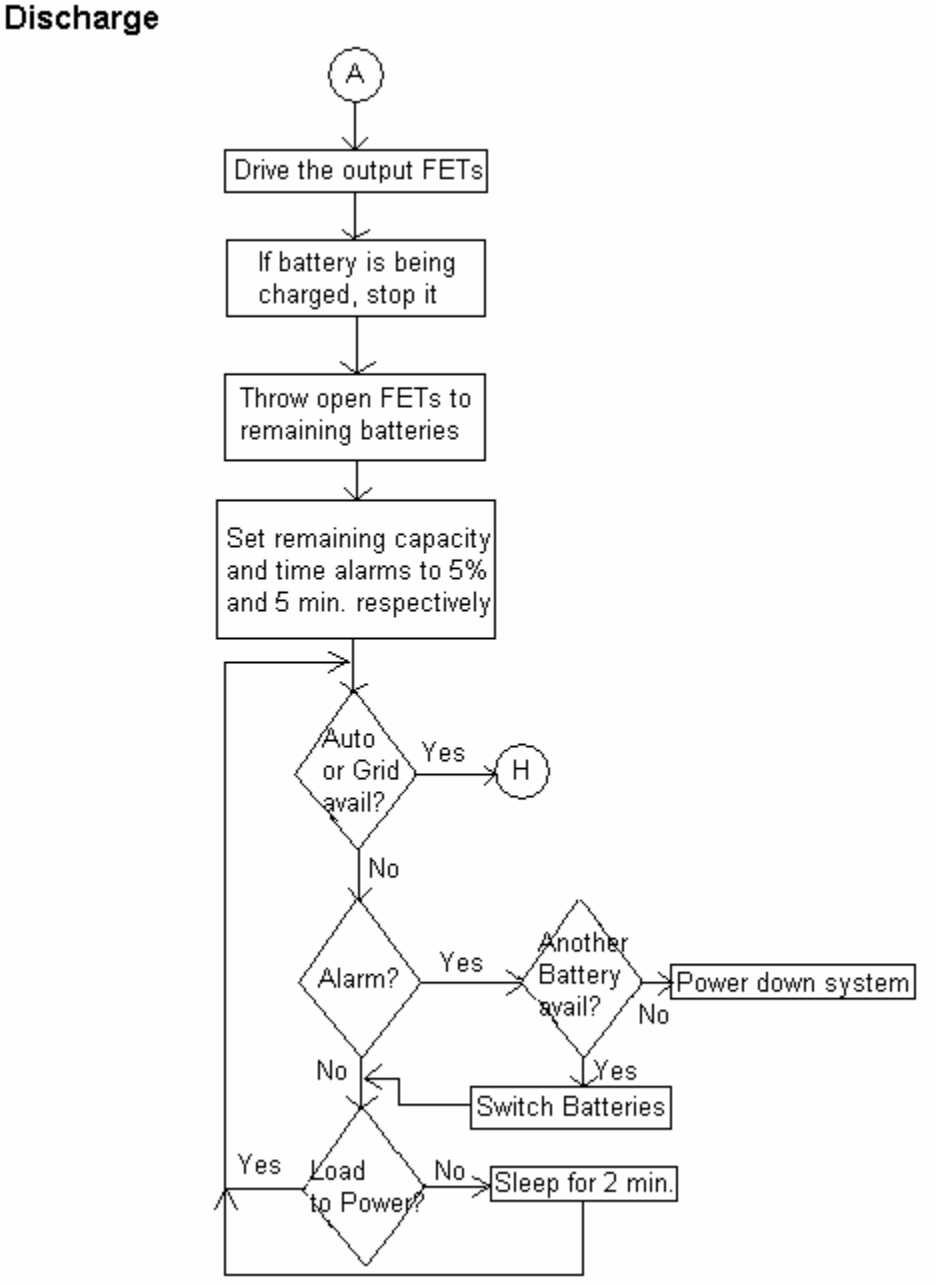

Figure 18. "Discharge" Flowchart

Next, we thrown open the FETs in the discharge paths of the remaining battery packs. We may already have opened some due to a battery possibly having a higher cycle count than others, but we want all FETs OFF for the remaining batteries because once the battery that is powering the system has discharged a bit, any of the remaining

batteries that still have the FETs ON would start powering the system as well due to the Schottky diodes. Once this is done, the following step is to set the remaining time alarm to 5 minutes and the remaining capacity alarm to 5%. As mentioned earlier, this is to get more power out of each of the batteries. After this, we then check to see if automobile or grid power is available. If it is, we will go to the "Auto or Grid" section of the program because there is no need to power the system with the batteries when auto or grid is available. Due to the Schottky diodes, we would not be able to use the batteries to power the system at this junction anyway. If auto or grid is not available, we continue on and check to see if any alarms have been thrown. We take care of the alarms in the procedure given above in the "Charge" section. We use the alarms to see when the battery is getting close to full discharg via the remaining time and capacity alarms. Because we are under the "Discharge" section of the program, when an alarm gets thrown by the battery, it will be something that causes us to switch to another battery. The battery to which we switch depends on which battery we are using to power the system. We switch according to the preference described earlier. In other words, if battery A1 is being used, we then use B1. If B1 is being used, we go to C1, and C1 goes to D1, and finally D1 to A1. If another battery is not available, then we have no power source to run the system, so we power down. If another battery is available we switch and continue on. Then we check to see if there is a load to power. We do this by checking the amount of current the battery pack is delivering. If it is delivering only a few dozen milliamps then we can be sure that the only load it is powering is the PIC18F452 microcontroller. If it is well above this, then we can assume that the imaging circuitry is requiring power. If there is no load to power besides the PIC, we go into sleep mode for a couple of minutes before continuing on. Sleep mode will shut down the main core of the microcontroller so we don't pull much current out of the battery, thus extending the life of the batteries. If there is a load to power besides the PIC, then we loop back and check to see if auto or grid is available. We will remain in the loop until auto or grid power is available. You may also notice that we do not check for charger alarms in this section. This is because the charger alarm we are concerned with affects charging, but we will not be charging while in this section because once again, we don't want one battery charging another.

If we have used all four batteries, and auto or grid power is not available, then the system will power down. This brings about the first improvement that could be made to this program. First, user-friendly code needs to be added so that users would know information they request such as which battery is powering the system or which battery is being charged as well as how much time the system can be powered by the batteries before the system shuts off. Secondly, it also needs to be mentioned that the preference used when switching from one battery powering the system to the next is not the optimal choice. If the batteries are used to power the system for as long as possible, that is until each is almost fully discharged, then this preference works fine. But if one or more of the batteries has a significantly higher charge/discharge cycle count that the others and the system is used for relatively short periods of time, then the batteries with more charge/discharge cycles will reach their end of life far before the others do. Therefore, the preference used should be determined by the number of charge/discharge cycles that each battery has. In other words, when switching to the next battery, it should be the one with the least number of charge/discharge cycles out of the ones available. This would allow the batteries with smaller cycle counts to catch up to the ones that have higher cycle counts. This wouldn't matter, however, if all four batteries are used every time that automobile or grid power is not available, because the batteries with higher cycle counts would reach end of life first anyway. The preference used in this program was chosen for simplicity.

A third improvement is if a battery does experience problems when charging or discharging, to take note of that battery and make sure it will not charge or discharge again until it is ready and able to. Even though the battery will likely not request charge if it has a critical condition, the program could still be improved by not allowing it to charge just in case it asks.

5.4 I²C Functions

Here, the I²C functions that are available are elaborated upon. The I²C functions that the PIC18F452 microcontroller provides are shown below in Table 2. The functions that are used in this program are AckI2C, CloseI2C, IdleI2C, NotAckI2C, OpenI2C,

ReadI2C, RestartI2C, StartI2C, StopI2C, and WriteI2C. AckI2C, NotAckI2C, RestartI2C, StartI2C, and StopI2C generate those conditions on the Bus as expected. Before we discuss the other functions, we will first mention the Master Synchronous Serial Port, or MSSP, module that is used for the serial communications. This module is capable of Serial Peripheral Interface communication, but we use the module for I²C, which is also supported.

The I^2C peripheral is supported with the following functions:

Function	Description
AckI2C	Generate I^2C bus *Acknowledge* condition.
CloseI2C	Disable the SSP module.
DataRdyI2C	Is the data available in the I^2C buffer?
getcI2C	Read a single byte from the I^2C bus.
getsI2C	Read a string from the I^2C bus operating in master I^2C mode.
IdleI2C	Loop until I^2C bus is idle.
NotAckI2C	Generate I^2C bus *Not Acknowledge* condition.
OpenI2C	Configure the SSP module.
putcI2C	Write a single byte to the I^2C bus.
putsI2C	Write a string to the I^2C bus operating in either Master or Slave mode.
ReadI2C	Read a single byte from the I^2C bus.
RestartI2C	Generate an I^2C bus *Restart* condition.
StartI2C	Generate an I^2C bus *START* condition.
StopI2C	Generate an I^2C bus *STOP* condition.
WriteI2C	Write a single byte to the I^2C bus.

Table 2. I²C Functions [6]

The ReadI2C function will read a byte from the Bus and the WriteI2C function will write a byte out on the Bus. But, before communication can begin, the MSSP module must be configured. This is done with the OpenI2C functions. There are two parameters within this function that can be used. The first determines whether the microcontroller will act as Master or Slave in the communication. Since we will always be initiating conversation to obtain data, which determines what actions we will take as the host, Master is always used when setting up the MSSP module. The second parameter determines whether the slew rate is enabled or disabled. If this parameter is

enabled, communication rates at 400 kiloHerz (kHz) are possible. However, because the maximum speed at which SM Bus-compliant devices can communicate is 100 kHz, we disable the slew rate and set up the communication speed at 100 kHz. The CloseI2C function disables the MSSP module, and it is used in this program at the end of functions that utilize the I²C functions, because there is no need to have the MSSP module busy when it is not needed. In addition, we use the IdleI2C function, which is very useful with systems that have more than one Master. When the Smart Battery wants to charge, it will continually broadcast its voltage and current requests to the charger chip. To do this, it acts as Bus Master. This is why we use the IdleI2C function. It monitors the Bus and makes sure there is no activity on the bus before we continue on in the communication protocol. This will help in preventing Bus collisions, which can be a problem in systems with multiple Masters.

5.5 Communication Protocol

The protocol taken when communicating with the devices in this system follow the protocol described above in Chapter 4. There are some things that must be added to the program when communicating, however, which will be explained shortly. The first step taken when communication begins is to use the IdleI2C function. This is necessary because any one of the batteries could be requesting charge, and we need to make sure we don't try to talk at the same time that they do. If we do, a Bus Collision will occur, which will scramble the data and cause the communication to be tried again later. It can also cause other problems as well. After IdleI2C says that the bus is clear, we then launch a timer which counts every microsecond (us) that goes by. This will be explained later.

Next we send the Start bit. After any condition is created on the Bus, we then must poll the SSPIF bit in the PIR1 register of the PIC18F452 microcontroller. When this bit is set, it means that the MSSP transmission is complete, which would be the Start bit in this case. After this, we need to clear the SSPIF bit, because that must be done in software, so that we can use the bit for later transmissions. Then we write out the address of the device we are trying to reach. After the write, we again must poll the SSPIF bit to

make sure we know when the write has completed. Also, a Bus collision function is created in the case that a Bus collision has occurred. The WriteI2C function returns a value after each time it is used. The value is zero if the write is successful, and it is -1 if a Bus collision has occurred. All we do in this function is to send a Stop condition to stop the communication and then clear a couple of bits. The bits are the WCOL bit in the SSPCON1 register and the BCLIF bit in the PIR2 register in the microcontroller. These bits get set when a bus collision occurs and must be cleared in software. After this, we then loop back to try communication again.

After we know that the write has been completed successfully, we check to make sure we get an ACK from the device we are talking to. This is done by checking the ACKSTAT bit in the SSPCON2 register in the microcontroller. This bit will be cleared if we receive an ACK and will be set if we get a NACK. If we get a NACK, then we start over and try the address again until we get an ACK. A NACK is usually sent when a device is busy, but this should virtually never happen in our system. Once we get an ACK, we continue on just as in the protocols listed above in Chapter 4, making sure we get an ACK after writes and polling the SSPIF bit after every write or condition is put out on the Bus to make sure that it is completed. Any time we poll the SSPIF bit, we start a timer. If this timer reaches 35 milliseconds (ms) before the SSPIF bit is set, then the device has timed out according to the SM Bus specifications [2]. Assuming that the device is fine, it is possible that this could happen if one of the devices has accidentally locked up the Bus. If a timeout occurs, we assume this has happened and clear the Bus. The Bus is cleared up by writing out FF's (we choose 8 bytes of FF's) on the Bus via the WriteI2C function. This should cause the device to finish clocking out any data it has, and then it waits for a Stop condition, so we then send out a Stop on the Bus, which should clear it up for communication. We then loop back to try communication again. In fact, any time we see a bus collision, get a NACK, have a timeout, or anything else that causes a communication failure, we will loop back and try communication again, after the problem has been fixed. This brings about another improvement that can be made. It is assumed that the device is fine and will eventually communicate with us. If it is not fine, we will be stuck in a communication loop forever. This can be fixed by having

another timeout detect that communication isn't working, and then assuming that the device is bad and not trying to talk to it again or use it in the system until it is replaced. Once again, the approach of assuming the device is fine is used for simplicity.

It needs to be mentioned that polling the SSPIF bit is a great way to run the communication protocol. Even though every device should be performing at 100 kHz, if some devices are not working up to speed, then it is okay because this system relies on the SSPIF bit, meaning that the device can talk slower than 100 kHz and be fine. The slower performance, of course must be within reason because timeouts will still occur if the bits aren't transferred within 35 ms.

5.6 Future Work

The imaging head and circuitry will need to be added as well as integrating the PIC18F452 microcontroller with the motherboard so that the PICDEM 2 PLUS DEMO BOARD will not be needed. Also, power consumption could be better considered in the design. The battery packs will consume much less current if in standby mode, rather than active. The problem is that if the clock or data lines are high, or at 5 Volts, then the Bus is considered active by the battery. The Bus will always be considered active in our system because the pull-up resistors cause the clock and data lines to always be high, which make the battery consume more current than necessary. Therefore, a future design of the system could float the clock and data lines down to zero Volts, and only allow the pull-up resistors to work when communication is needed. This would cut down on the power wasted a great deal when automobile or grid power is available and we don't need the batteries to be in use. Furthermore, due to the location of the battery connectors on the charger boards, the batteries will slip off of the connectors due to the batteries' weight. The charger boards need to be re-designed so that the connectors are lower, allowing the batteries to rest on the same surface that the motherboard is resting on, rather than having to put something, such as a book, under that batteries to compensate.

Hot swapping, in this case, is pulling out one of the battery packs and replacing it with another while the system is still running. The current design is not suited for this application, but a future design should allow hot swapping to make the system more

robust. Additional circuitry must be added to allow hot swapping, and the program must be modified as well.

To recap, the following changes need to be incorporated into future designs of the system. For the charger boards, the cathode of diode D4 needs to be connected to the cathode of diode D5. Make sure that capacitor C3 has a value of 47nF and not 0.47nF. Capacitor C6 is actually made up of two capacitors in parallel, so that extra capacitor needs to be designed into the system. Because the thermistor value in the NI2020 Smart Battery and the thermistor value that the MAX1667 charger chip expects to see are different, circuitry needs to be added to the charger board that will prevent too much current from being drawn out of the REF pin (#9) on the MAX1667. This way a 300 Ω resistor can be used for R3 to match the 300 Ω thermistor in the NI2020 Smart Battery. For the motherboard schematic, the grid power connector needs to be re-designed so that +15 V will reach the motherboard, rather than -15 V. The last change that needs to be made, but was not done during the test phase is the load switch or output FET bank. The type of FETs used need to allow full voltage from any of the power sources to get through to the load when turned ON, instead of 2 V less than full voltage, which is what the current FET bank will deliver to the load when ON.

In software, there are several things to be done to improve the current program. First, user-friendly code should be added to alert the user of critical conditions and to display information that the user may want to know. Next, the preference in determining the battery pack to power the system after another is depleted could be improved. Also, not allowing a battery that has recently had a critical condition to either power the system or charge is a better answer than the design currently used. The last improvement is using the RelativeStateOfCharge() register in the Smart Battery to determine when a battery needs to be charged. Running full charge/full discharge of the battery packs will provide longer life for the packs and in turn the system, and the current design does not guarantee full charge/full discharge of the battery packs in all situations.

REFERENCES

(Reference #1 is the paper that Hari P. Nayar wrote for the hardware design of this system. The remainder of the references are datasheets or specifications, downloaded as PDF files from the different companies' websites.)

1. Nayar, Hari P. "Design of Intelligent Power Module for Portable Colposcope," Masters Thesis, Texas Tech University, 2004.

2. Duracell/Intel. "System Management Bus Specification." 1995, www.intel.com.

3. Inspired Energy, Inc. "Battery Specification – NI2020A22." 2003, www.inspired-energy.com.

4. Maxim, Dallas Semiconductor. "Chemistry-Independent, Level 2 Smart Battery Charger – MAX1667." 1999, www.maxim-ic.com.

5. Philips Semiconductors. "The I²C-Bus and how to use it (including specifications)." 1995, www.philips.com.

6. Microchip. "MPLAB C18 C Compiler - Libraries." 2003, www.microchip.com.

7. Analog Devices. "CMOS, Low Voltage, 2-Wire Serially-Controlled, Matrix Switches – ADG728/ADG729." 2000, www.analog.com.

8. Duracell/Intel. "Smart Battery Data Specification." 1995, www.intel.com.

9. Microchip. "I²C." 2001, www.microchip.com.

10. Microchip. "PIC18FXX2 Data Sheet." 2002, www.microchip.com.

11. Microchip. "MPLAB C18 C Compiler – Getting Started." 2003, www.microchip.com.

12. Microchip. "MPLAB C18 C Compiler – User's Guide." 2003, www.microchip.com.

13. National Semiconductor. "LM2940/LM2940C 1A Low Dropout Regulator." 2003, www.national.com.

APPENDIX
CD-ROM CONTENTS

The CD-ROM contains the program that was written to manage the system described in the paper, as well as several test programs that were written to test individual functions of the system. The programs were written in C and tested using the Integrated Development Environment (IDE) provided in MPLAB IDE v6.60 from Microchip. The main program was never completely tested due to financial restrictions (only had two Smart Batteries to test), so the program was modified as if the system only used 2 Smart Batteries.

PERMISSION TO COPY

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Texas Tech University or Texas Tech University Health Sciences Center, I agree that the Library and my major department shall make it freely available for research purposes. Permission to copy this thesis for scholarly purposes may be granted by the Director of the Library or my major professor. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my further written permission and that any user may be liable for copyright infringement.

Agree (Permission is granted.)

Chris Caceres	3/11/05
Student Signature	Date

Disagree (Permission is not granted.)

_____	_____
Student Signature	Date