Programmable LED Marquee

COMP SCI 353 Computer Architecture and Organization Project
Fall 2006

Daniel Kelly

Table of Contents

1. Motivation

My motivation for this project came from a couple of different places. I have always wanted to build one of those LED displays that can be seen in front of banks and stores. In one of my classes this semester, the professor asked everyone to make a name-card so he could learn their names more quickly. I made one that lit up by hot-gluing some LED's to the back and hooking them up to a small battery pack. I wished I could make an actual sign then, but I didn't think it was feasible. Then when the opportunity came up to do an implementation project using a microprocessor, I thought, Hmm… maybe I could build one of those signs as part of this project. It seemed like a logical choice. After all, we were going to study logic and the TK-Gate seemed like a good way to simulate such a design. I didn't really think at first that I would be able to pull it off, but at the very least I could figure out what it would take and maybe make one sometime in the future after school was done and I had more time & money on my hands. One other thing that motivated me toward this project was that I had recently come into a couple 5x7 dot matrix segments, (which proved to be the wrong kind, but that's another story.)

2. Research Conducted

The first thing I did after getting my proposal accepted, was to figure out what I had on hand for parts. I dug out all my old electronics hobby stuff from high school, and made an inventory of it. I also built a circuit implementing the traffic light scenario just to 'get my hands dirty' again. Then I did a lot of research on the Internet looking for different ways to build LED matrix signs. I found a lot of interesting ideas, but not any real easy implementations that I could adopt. My

reasoning is that since these signs are commercially quite expensive, their builders must keep their construction a closely guarded secret. I had a 6502 microprocessor, and some RAM chips from out of an old PC, and I thought about trying to use them in my design. I found some interesting articles and web sites about that old chip, but they seemed to imply that I would need to buy and program an EEPROM, which I didn't really know how to do. In the course of my research I also found that a couple of recurring themes.

Dot Matrixes are controlled not by turning on a bunch of LED's at once in a pattern, as one might think intuitively, but rather buy turning on a single LED or a column of LED's in a pattern, and then turning on the next column, and the next, with only one column at a time being lit up. Then after lighting all the different columns (either from left to right, or right to left), repeat the process, but very fast so there is no flickering. The entire process is similar to the way a television CRT screen works.

Another common idea was the use of a microprocessor to store the patterns, and then using shift registers to carry the pattern down to each LED matrix segment. Of course no one specifically spelled out how either thing was to be done, but at least it gave me an idea for a starting point.

Finally, I should mention the Maxim 6592 chip. They cost anywhere from $8 to $15, but I was able to acquire one for free by requesting a sample from the Maxim website. It seemed like the perfect solution at first because it had an entire ASCII font set stored in ROM, and could be accessed by a microprocessor serially, and could control up to 4 LED matrix displays simultaneously. Unfortunately, the first thing I did with it was hook it up backwards, which I think might have damaged it. I

spent 3 days trying to get my microprocessor to talk to it, but finally had to give up and pursue my own method of creating patterns and driving the displays. It was a good learning experience because even though it didn't work. I learned about SPI (serial peripheral interface) communication, which is a method for embedded devices to communicate using a 4-wire synchronous protocol that is widely used in the electronics field. I will definitely be trying to make it work again after this project is over. Maybe I can get another free sample.

Eventually my research led me to Microchip, who makes the PIC family of micro controllers. They have a relatively inexpensive ($60) development kit called the PICkit 2 Starter Kit, which I decided to buy after thoroughly investigating it on their website. I also bought some extra microprocessors in case I damaged one, and I was glad I did because it let me create new programs without having to overwrite the old ones. The starter kit came with a demo board and 12 lessons that turned out to be just enough to do teach me what I needed to do this project. I wish the lessons had gone up to SPI, but I guess that was considered too advanced a topic for a starter kit.

The most useful research materials for this project turned out to be the datasheets for the individual devices. The Microchip website had a technical forum, but their site was so complex, that I wound up going to back to the datasheet over and over instead.

I don't believe I will be quoting any particular web site, since my project was mostly built by trial and error and by using the datasheets, but I will include all the links that I put in my browser bookmark within my references section. They deserve some credit for pointing me in the right direction, and they may prove valuable to

anyone wanting to pursue a project of this type on their own. I will organize them by the categories of Interesting, Useful, and Dead-Ends. I am also going to include the datasheets themselves in my appendix, so the reader doesn't have to go searching for them.

3. Technology

I will briefly describe the technology used in this implementation instead of examining it in detail for two reasons: First, the focus of this paper is how the technology is used to implement the LED sign, not the specific technology itself. Second, because the datasheets are so detailed, it is almost impossible to describe the devices in detail without just reproducing that information. If the reader desires to delve deeply into how the individual devices work, links to the datasheets and manuals are included in the appendix.

3.1 PICkit 2 Starter Kit and Low Pin Count Demo Board

The most important piece of technology was the PICkit 2 starter kit. The best description of it can be found on their website, as shown below:

PICkit™ 2 Starter Kit **DV164120** (purchase from Microchip Direct)

The PICkit™ 2 Starter Kit is a low-cost development kit for programming many of Microchip's baseline, midrange, PIC18F, PIC24, and dsPIC33 families of Flash memory microcontrollers. This starter kit is designed to help the developer get up to speed quickly using PIC® microcontrollers. The kit provides everything needed to program, evaluate and develop applications using Microchip's powerful midrange Flash memory family of microcontrollers. Instructions are provided in a series of twelve lessons that cover I/O, interrupts, A/D converters, data tables and timers. All source code files for the lessons are furnished.
The Low Pin Count Demo board supports all PIC12F and PIC16F 8/14/20 -pin products.

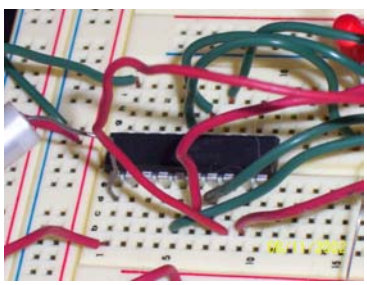
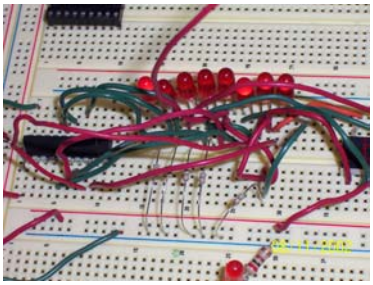
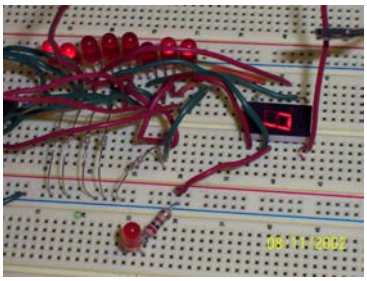
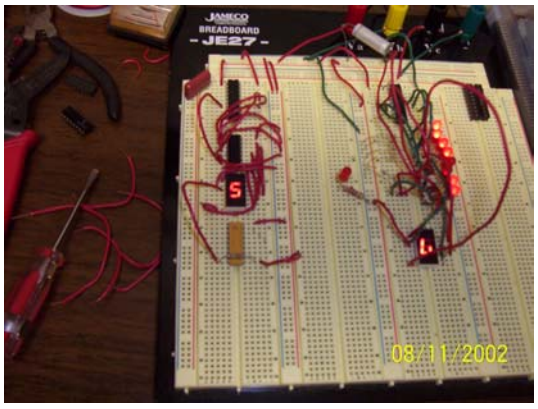
Features of PICkit 2 Starter Kit

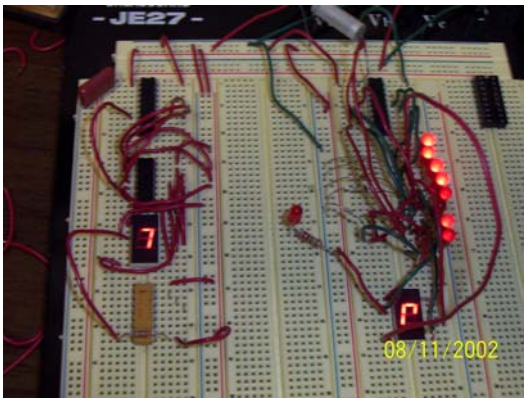
- Low pin count demo board supporting 8/14/20-pin mid-range PICmicro microcontrollers
- Easy to use Windows® programming interface for programming many of Microchip's Flash family of microcontrollers
- Twelve sequential lessons written in assembly code demonstrate how to use Microchip's midrange Flash family of microcontrollers
- Microchip's Tips 'n Tricks Booklets provides efficient, low-cost design techniques using Microchip Flash microcontrollers
- PICkit 2 User's Guide (included on CD ROM)
- Low Pin Count Demo Board User's Guide
- FREE! Microchip's **MPLAB IDE** software for a complete code development environment
- FREE! **HI-TECH PICC™ LITE C Compiler** (contained on the MPLAB CD)
- FREE! **CCS PCB™ Baseline C Compiler** (contained on the MPLAB CD)

I found the PICkit extremely easy to use, and went through all 12 of the lessons in one day. Making my own programs with the included MPLAB software proved a little a little more tricky because I kept overwriting my original projects, but eventually I figured it out. I highly recommend it as a way to learn how to develop embedded applications.

Once I got my first program working, I re-built it on a solder less breadboard to test, and got that familiar feeling that programmers get when their first program works in a new language. Similar to how a proud father feels when his kid says his/her first word.

The first program I worked with was called rotate. It placed a binary value into a file register, and then shifted it one bit left, with a carry bit coming back around. It used the 4 LED's on the demo board. I modified it to use 8 LED's (one for each bit), and sent it a value so 4 of them were lit at a time. I then used it to experiment with lighting segments of a 7-Segment display. I needed pull-up resistors as shown in the photo to keep the active low outputs from floating as shown below:



Unfortunately, I overwrote the program before I fully understood how to save projects in MPLAB. I also had to pull apart the circuit to make room for the next development phase. Luckily, because I ordered 5 extra microprocessor chips, I kept the programmed IC, and the photos, so I can hook up the circuit again.

3.2 The Mighty 16F690 Microprocessor

The 16F690 is a 20-pin, Flash-Based CMOS Microcontroller. It uses the RISC architecture like we learned about in class, with 35 instructions. The instruction set and block diagram are shown below:

Instruction Set:

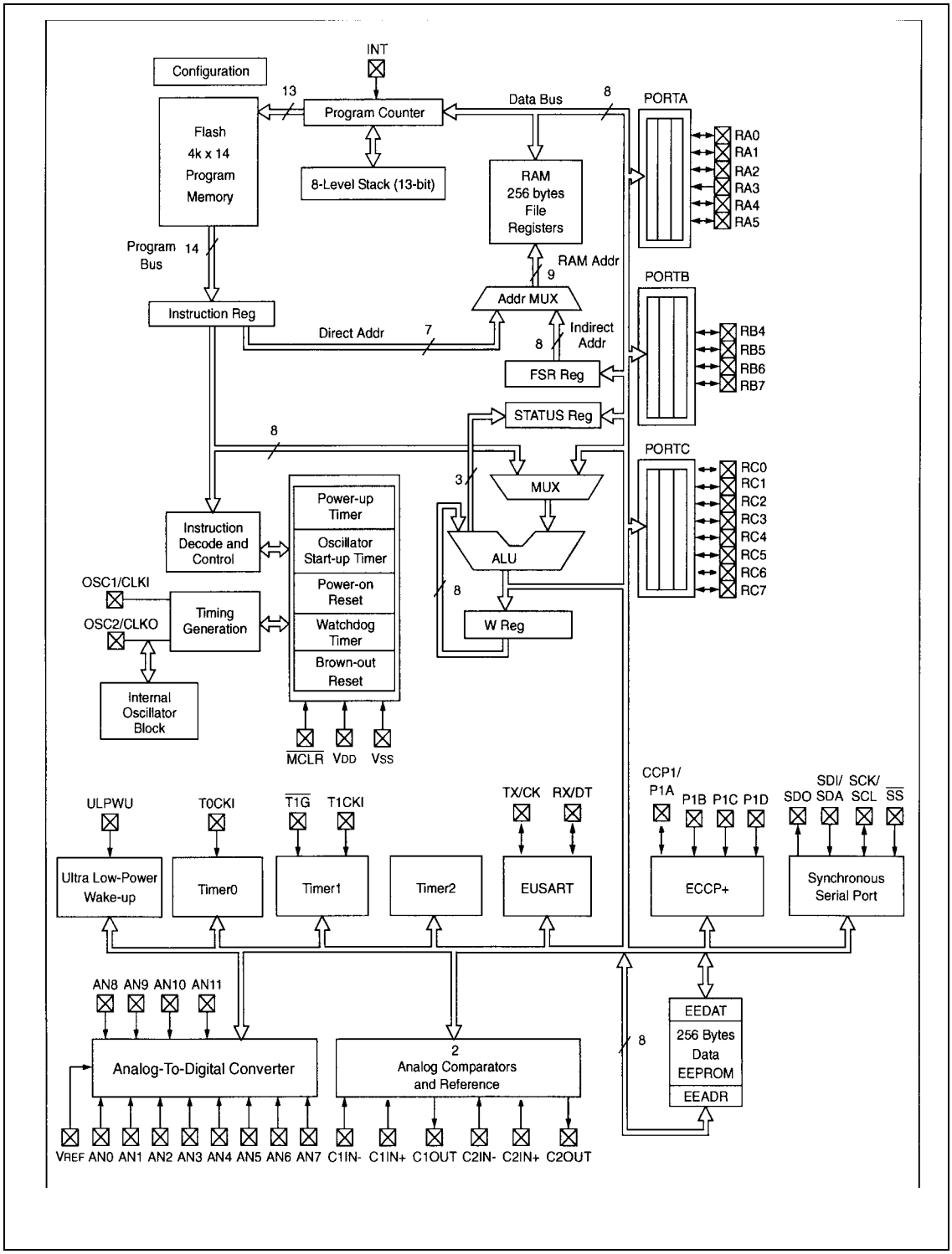
Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS							
ADDWF	f, d	Add W and f	1	00 0111 dfff ffff		C, DC, Z	1, 2
ANDWF	f, d	AND W with f	1	00 0101 dfff ffff		Z	1, 2
CLRF	f	Clear f	1	00 0001 lfff ffff		Z	2
CLRW	–	Clear W	1	00 0001 0xxx xxxx		Z	
COMF	f, d	Complement f	1	00 1001 dfff ffff		Z	1, 2
DECF	f, d	Decrement f	1	00 0011 dfff ffff		Z	1, 2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00 1011 dfff ffff			1, 2, 3
INCF	f, d	Increment f	1	00 1010 dfff ffff		Z	1, 2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00 1111 dfff ffff			1, 2, 3
IORWF	f, d	Inclusive OR W with f	1	00 0100 dfff ffff		Z	1, 2
MOVF	f, d	Move f	1	00 1000 dfff ffff		Z	1, 2
MOVWF	f	Move W to f	1	00 0000 lfff ffff			
NOP	–	No Operation	1	00 0000 0xx0 0000			
RLF	f, d	Rotate Left f through Carry	1	00 1101 dfff ffff		C	1, 2
RRF	f, d	Rotate Right f through Carry	1	00 1100 dfff ffff		C	1, 2
SUBWF	f, d	Subtract W from f	1	00 0010 dfff ffff		C, DC, Z	1, 2
SWAPF	f, d	Swap nibbles in f	1	00 1110 dfff ffff			1, 2
XORWF	f, d	Exclusive OR W with f	1	00 0110 dfff ffff		Z	1, 2
BIT-ORIENTED FILE REGISTER OPERATIONS							
BCF	f, b	Bit Clear f	1	01 00bb bfff ffff			1, 2
BSF	f, b	Bit Set f	1	01 01bb bfff ffff			1, 2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01 10bb bfff ffff			3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01 11bb bfff ffff			3
LITERAL AND CONTROL OPERATIONS							
ADDLW	k	Add literal and W	1	11 111x kkkk kkkk		C, DC, Z	
ANDLW	k	AND literal with W	1	11 1001 kkkk kkkk		Z	
CALL	k	Call Subroutine	2	10 0kkk kkkk kkkk			
CLRWDT	–	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$, $\overline{PD}$	
GOTO	k	Go to address	2	10 1kkk kkkk kkkk			
IORLW	k	Inclusive OR literal with W	1	11 1000 kkkk kkkk		Z	
MOVLW	k	Move literal to W	1	11 00xx kkkk kkkk			
RETFIE	–	Return from interrupt	2	00 0000 0000 1001			
RETLW	k	Return with literal in W	2	11 01xx kkkk kkkk			
RETURN	–	Return from Subroutine	2	00 0000 0000 1000			
SLEEP	–	Go into Standby mode	1	00 0000 0110 0011		$\overline{TO}$, $\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11 110x kkkk kkkk		C, DC, Z	
XORLW	k	Exclusive OR literal with W	1	11 1010 kkkk kkkk		Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF GPIO, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

2: If this instruction is executed on the TMR0 register (and where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 module.

3: If the Program Counter (PC) is modified, or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

Block Diagram:



The 16F690 uses separate program and data memory areas. The data is held in file registers using 7 bits, and multiple file registers are arranged into pages, which are accessed by two extra bits in the status register. I had one interesting code problem early on because I was accessing the wrong page, even though I had the correct address. Once I noticed that, the code started working properly. There are 32 addresses called Special Function Registers (SFR) that allow the device to interact with peripherals. The controls and data registers are mapped directly. For instance, TRISC 0 will set the control bits for port C to be outputs, and then PORTC 11111111 will set the data bits to 1 for each bit in the SFR. The addresses above 0x20 to the end of each page are General Purpose Registers (GPR) where program variables can be stored. Program memory is accessed with a 13-bit Program Counter (PC). It works similarly to the PC in the MIPS data path discussed in class.

Instructions are Byte oriented, Bit oriented, or Literal. For Byte instructions, there is a 7-bit address, a destination bit, and a 6-bit op-code. One operand is made of the data address plus the 2-bit page address, and the other is the working register (W or Wreg). The destination bit determines where the result will be stored (in W, or in back in the original file register). For Bit instructions, there are 7 bits of data, a 3-bit number, and 4 bits of op-code. They can test, set, or clear a specific bit in a file register. For Literal instructions, the data operand is contained in the instruction, and the other operand is Wreg.

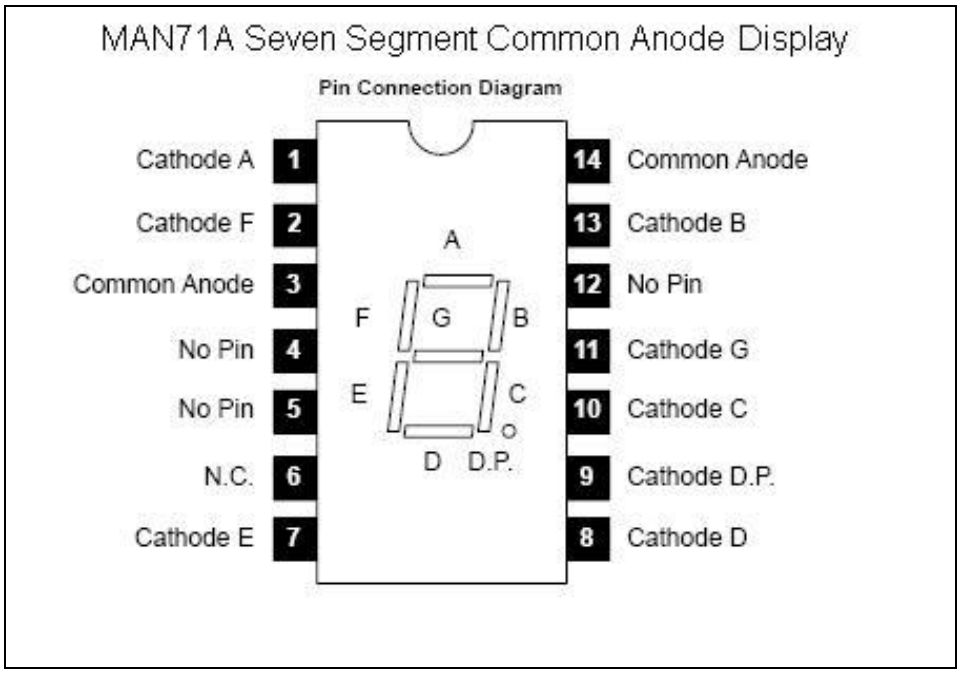
Other than the fact that it is an 8-bit microcontroller, instead of 32-bit like we used in class, programming the code in the assembler was just like using PCSPIM for the

MIPS instruction set. It just has smaller instructions to figure out. In fact, the differences between what I did to program this chip, and what we did in class programming with the MIPS architecture are really only semantic differences similar to writing the same programs in Java and 'C'. I don't want to get into detail any deeper here, as it is all explained much better in the 292-page datasheet for the chip. I will, of course, explain each of the instructions that I used in the final code example by using comments in the code.

3.3 7-Segment Displays vs. Dot Matrix Displays

The seven segment displays are common anode, which meant that I had to send the opposite pattern of bits to light them. For instance, if I wanted all segments to light, then I had to send all zeros in the bits, instead of all 1's.

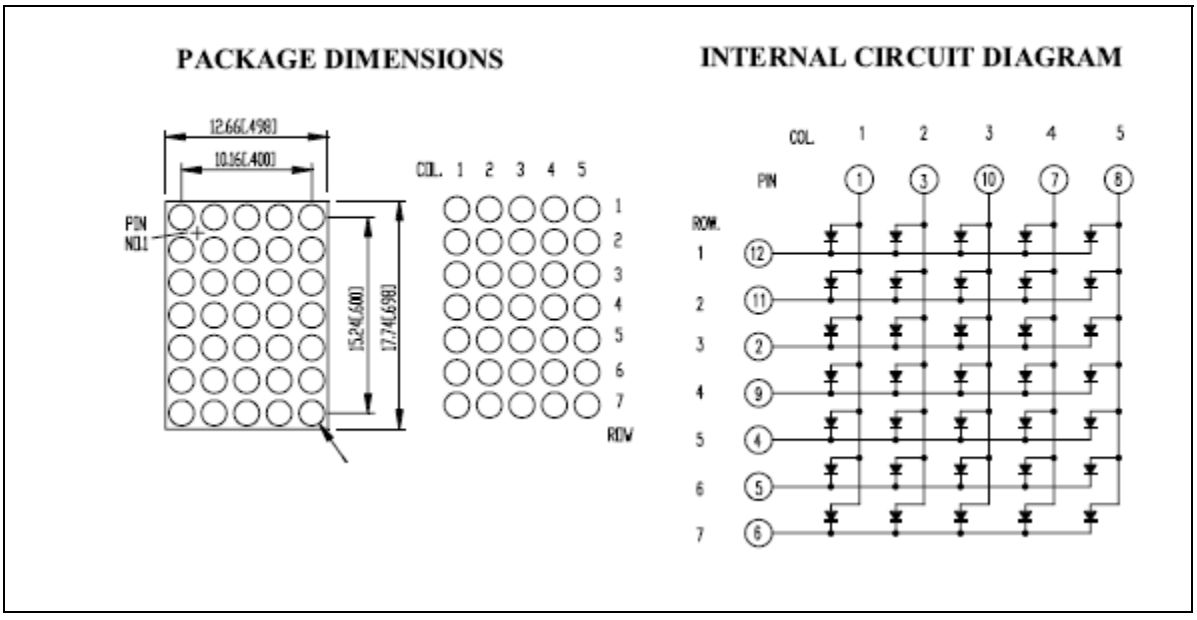
Seven segment Display



I controlled the seven segment displays two different ways. In the Countdown program, I sent a 3-bit binary number to the 7447 BCD (Binary Coded Decimal) decoder chip, which uses combinational logic to decode the binary number and light up the appropriate segments on the display. In the 7-Segment Message program, I directly controlled the segments of the display by setting the lower 7 bits on PORTC, and using the MSB on PORTC as a clock signal. I sent the clock signal and each bit to its own 8-bit Serial In, Parallel Out (SIPO) shift register, which meant that the character being sent could shift from one display to the next.

The dot matrix displays are common cathode rows and common anode columns. That means to turn on an LED, I have to send a low (0) to each of the desired the row pins, and a high (1) to the desired column pin.

Dot Matrix Display

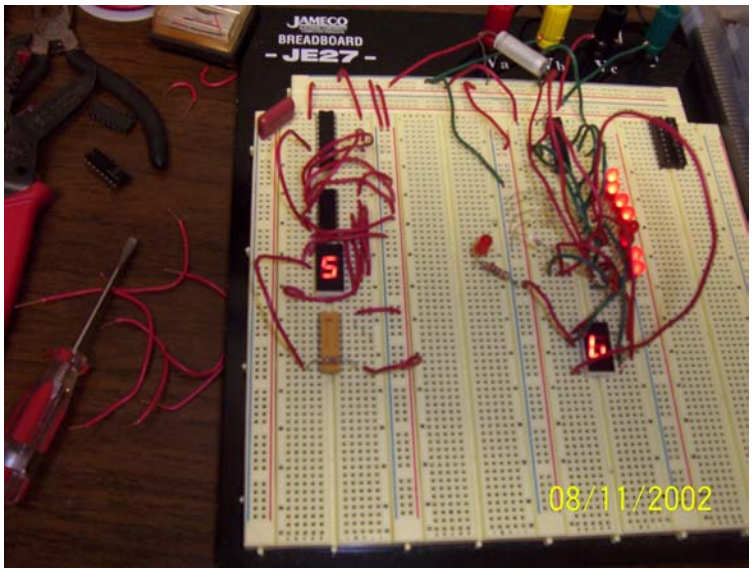


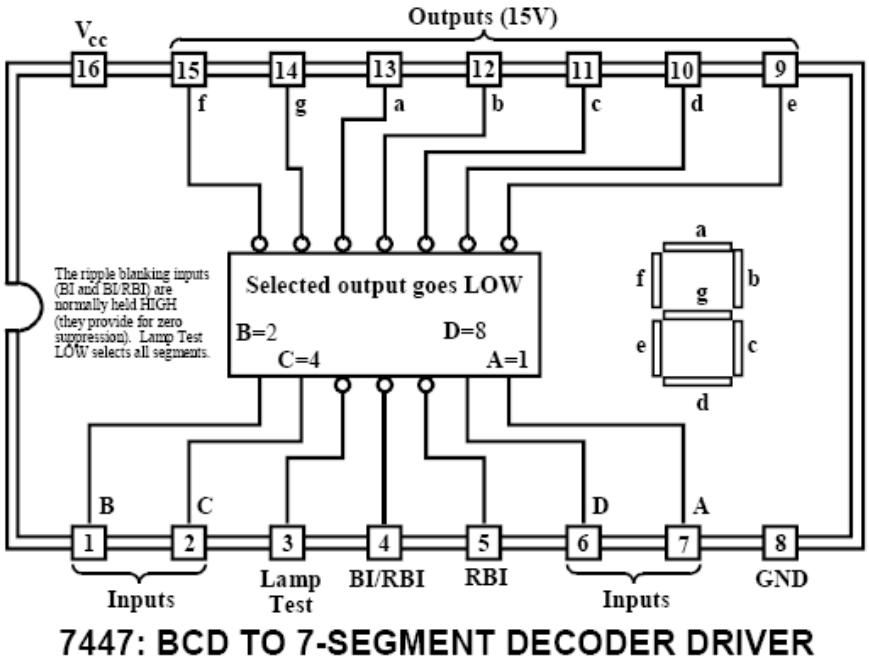
In my Dot Matrix program, I used the seven lower bits of PORTC of the microprocessor to control rows 1 through 7 as active lows (0=on), which I wired in

parallel for all five displays. I used the two upper bits of PORTB as active highs (1=on). I used one bit for the clock signal, and one to light up the first column. Then I clocked that one column bit through the four shift registers, changing the row bits each time so that only one column at a time lights up. The algorithm is shown below:

3.4 BCD Decoder

The 7447 operation is pretty much straight forward. It takes 4 binary inputs and turns on outputs (active low) corresponding to the seven segments of the display. For instance, if I send 0101 to inputs A, B, C, and D it will send lows to segments a, f, g, c, and d, which will light up as 5 as shown below:



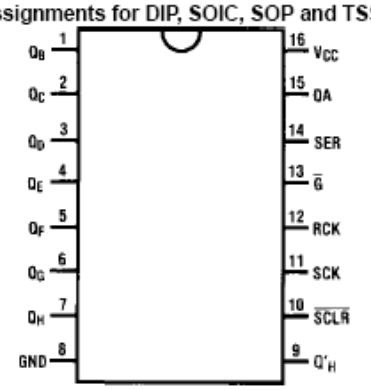


3.5 Shift Registers

The shift registers are 74HC595 8-bit Serial-In, Parallel-Out (SIPO) that includes a D-Type storage register.

Connection Diagram

Pin Assignments for DIP, SOIC, SOP and TSSOP

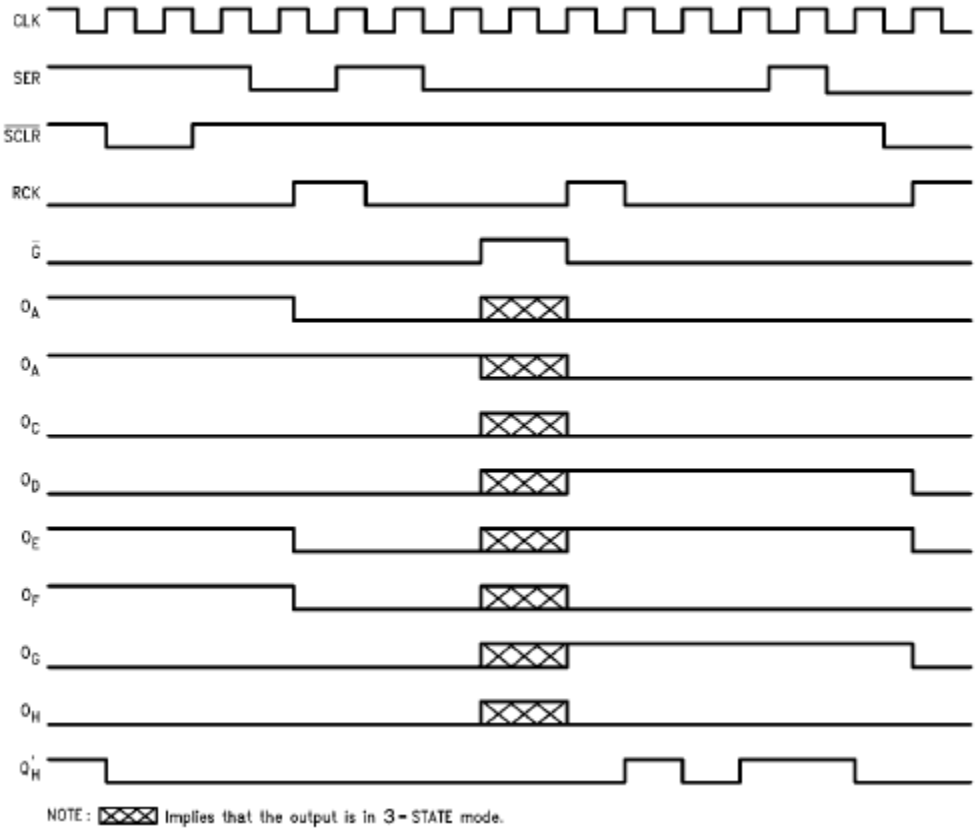


Top View

Truth Table

RCK	SCK	$\overline{SCLR}$	$\overline{G}$	Function
X	X	X	H	Q_A thru Q_H = 3-STATE
X	X	L	L	Shift Register cleared $Q_H = 0$
X	↑	H	L	Shift Register clocked $Q_N = Q_{n-1}$, Q_0 = SER
↑	X	H	L	Contents of Shift Register transferred to output latches

Timing Diagram



NOTE : ⊠ Implies that the output is in 3-STATE mode.

I tied the clock signal for both the serial register and the storage register together so that the storage register is always one pulse behind the shift register. It has an output Q'h that allows for multiple shift registers to be linked together in series, so in my circuit it is like having a 32-bit SIPO. On the 7-segment display circuit, each register has a segment of the first display for input, and then each output is that same segment, but on sequential displays. On the first clock pulse, the first display is lit with data sent from the microprocessor, and shift registers 'a' through 'g' are loaded with the first serial data. Initially all shift registers are empty, so displays show '8' (all lows). Then on the next clock pulse, the first display is lit with new data, and the shift register loaded with new data, and the old data is sent to the next storage bit in each shift register, lighting display 2 with the data formerly in display 1. This continues with each clock pulse.

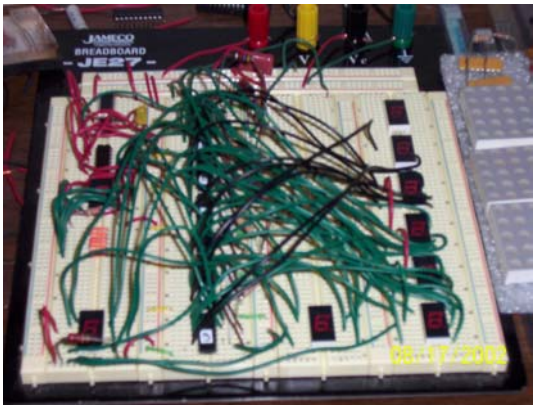
In the dot matrix example, the shift registers are to turn on the columns of the common anode displays. A start bit is sent when the program cycle starts, and then it is clocked sequentially to light up each column. After a word (5 characters) has been displayed, a new start bit is sent to start displaying the word again. The entire cycle happens many times very fast to display the word, many times per second. To display multiple words, I set a loop and counter in place. For instance 'Hello' is drawn 0xFF times, then 'World' is drawn 0xFF times. Then the whole program loops indefinitely.

4. Design Phases Overview

I decided to build the display in three phases in case something went wrong and I couldn't finish on time. I got this methodology from software development and program management classes. It helps keep the project from sliding out of scope as well. The first phase was to get it the microprocessor programmed and working on the breadboard, and to make a single 7-Segment light up and display something. Next I wanted to hook up several displays with the shift registers and make sure that it was possible to cascade the characters in the shift registers. Finally, I would attempt to combine those two ideas, and use the dot matrix displays so that I could make the entire ASCII character set, or any characters that I wanted. Phase 4 would have been to try and do it with the Maxim 6952 chip, but that proved to beyond the timeline of the project.

5. Phase 1: 7-Segment Countdown Program

I used the rotate program to test polarity and values of resistors, as well as functionality of the chip on the breadboard as opposed to the demo board. Then I made a countdown program using the BCD decoder to make a simple program that counts down from 9 to 0. Here are some photos of the proto board running at phase 1.



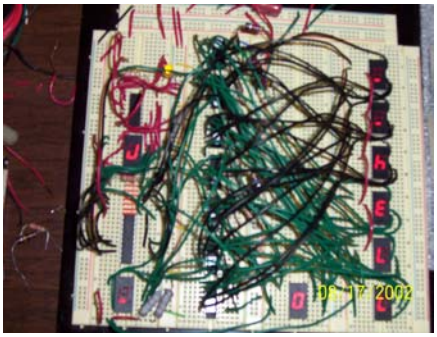
More photos and videos of the programs running can be found on the CD-ROM in Appendix E.

The commented code can be found in Appendix A. The algorithm for the program is shown below:

1. Create a variable 'Display' and set its value to zero
2. Load decimal value 9 into working register (Wreg)
3. Move the Wreg value to 'Display'
4. Perform some delays using the internal timer
5. Move the 'Display' value to the working register (Wreg)
6. Move the Wreg value to PORTC (the I/O pins)
7. Decrement the 'Display' register, and test if Display = zero
   If Display=zero, then do not branch, else branch to 4.
8. Move value decimal 9 into Wreg
9. Move Wreg into 'Display'
10. Go To Step 4

6. Phase 2: 7-Segment Message Program

The 7-Segment Message Program bypasses the BCD chip and instead directly turns the LED segments on and off. In addition to sending the 7 bits to the display, it also sends them to seven shift registers. It also sends a clock bit to all the shift registers. Here are some photos of the 7-Segment Message Program.

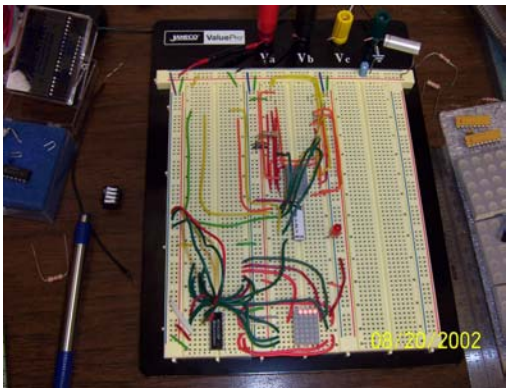


Additional photos and videos of the programs running can be found on the CD-ROM in Appendix E. The commented code can be found in Appendix A. The algorithm for the program is shown below:

1. Create variables for 'Display' and 'Delay1' and 'Delay2'
2. Set PORTC to outputs
3. Set 'Display' to zero
4. Move literal data into Wreg (display bits ie. '01110111')
5. Move Wreg into 'Display'
6. Move 'Display' into Wreg
7. Move Wreg into PORTC
8. Call Delay Routine
9. Move 'Display' into Wreg
10. Add '10000000' to Wreg (sets clock bit high)
11. Move Wreg to PORTC
12. Call Delay
13. Repeat steps 4-12 for each character
14. Go to Step 4 forever

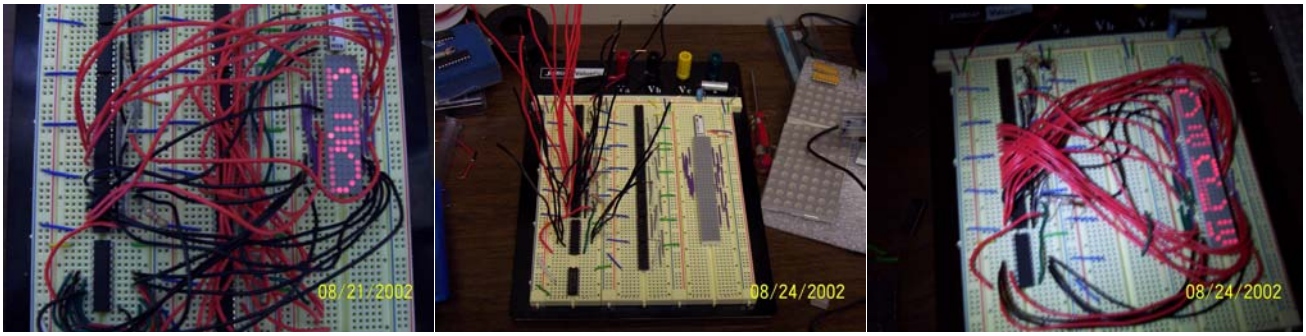
7. Phase 3: Dot Matrix Display Programs

I finally was ready to start programming the 5x7 dot matrix display programs. First I did some electrical testing to see which polarity the displays were and to match them up to the datasheet drawings.



I tried to do a complex circuit involving sending the columns to one bank of shift registers, and the rows to another bank of them. At first, it seemed to work, but then I realized that I was only able to get the same character on every display that way. For instance, the best I could manage was to get it to say 'DDD' pause 'AAA' pause 'NNN'. That would never do,

and I almost gave up, when I realized that what I really wanted to do was wire the displays up as though there was only one very long display (instead of 5 columns by 7 rows, I should think of them as 25 columns by 7 rows). It actually wound up being a much cleaner wiring and program than what I had originally thought.



Then I could use the same code and schematic for one display, and copy-paste the code for each successive character, looping back to the beginning after five characters to repeat the loop and make 5-letter words at a time.

Using this I could write 'HELLO', or any other 5-letter or less word, but how to make a second word? The answer was to insert a counter so that the first word displays a certain number of times (I chose hex number 0xFF), and then the second word loops the same number of times, and then the third, etc. At the end just loop the program back to the beginning and start over again. Finally I had my 'Hello World' program. I am still working on making it scroll, but I had to stop at this point because I am out of time. Here are some photos of the Hello World Program running:

Additional photos and videos of the programs running can be found on the CD-ROM in Appendix E. The code for the Hello World program can be found in the appendix and the algorithm for the program is below:

1. Turn off all bits on PORTB and PORTC
2. Turn on Data Bits (PORTC set bits 0 through 6, 0=on, 1=off)
3. Turn on Start Bit (PORTB set bit 6 = 1)
4. Turn on Clock Bit (PORTB set bit 7 = 1)
5. Turn off Clock Bit (PORTB set bit 7 = 0)
6. Turn off Start Bit (PORTB set bit 6 = 0)
7. Turn off Data Bits (PORTC set all bits 0 through 6 = off)
8. Turn on Clock Bit (shifts data to next column)
9. Turn on Data Bits (PORTC set bits 0 through 6, 0=on, 1=off)
10. Turn off Clock Bit
11. Repeat steps 7-10 for each character
12. Go to Step 2 to re-draw word (word = 5 characters)

8. Conclusion

I learned a lot from doing this project. I also can honestly say this was the most fun project I've worked on in 6 years at UWGB. I didn't think I was learning much in this class that was relevant at first, but when I started trying to figure out the data sheets it occurred to me that I would not have been able to decipher them without the knowledge that I've gained this semester. It became even more apparent when I tried to explain to some of my friends at work how I was able to program the microprocessor. When I talked about setting data and control bits in the various registers and they gave me a blank stare, I realized that something I had no knowledge of 3 months ago had become as fluent as a second language to me within a small time. The most exciting thing of all is that an area of my life that I thought I

would never get back into again has become re-opened. I am already planning what else I can do with these mighty little microprocessors.

The next phase that I plan to explore with the LED project is working with the input side of the I/O. I want to add a keypad so that I can program in the characters as a table, and then set which ones to display on the fly, without having to re-burn the chip each time. I think I can use an old telephone keypad as the input device. Basically it will turn it into a completely stand-alone display. After that, I have some stepper motors that I removed from various printers over the years and I'm just itching to build a robot or two. From there, the sky is the limit. One very practical device that comes to mind is a digital speedometer for my jeep, and maybe I can hook up a thermal sensor and make a digital thermostat for my furnace.

Another thing I would like to do is design a trainer or demo board of my own for hobbyists and students interested in using a PIC microcontroller. There are only a couple good books out there, and the average person without a formal education in this field would find it overwhelming to go through the datasheets the way I did. I think I could write a fairly good manual about how to design and build circuits like the ones I am going to use these little chips for. In addition it would fill a growing hobbyist niche market in the robotics field. I could put it on my website and try to sell it as a startup kit for a reasonable price.

9. References

Interesting

http://www.makezine.com/blog/archive/2005/05/5x7_led_dot_mat.html

http://www.csee.wvu.edu/~cukic/CPE181/New_cpe481/group17/manuals/servicemanual.pdf

http://cs.nyu.edu/~jhan/ledtouch/index.html

http://ledlightray.com/tag/led-matrix/

http://www.hanssummers.com/electronics/clocks/matrix/

http://www.slscorp.com/pages/ledsls.php

http://www.ugcs.caltech.edu/~bret/create_index.html?create_electronics_ledmatrix

http://ask.metafilter.com/mefi/35456

Useful

https://itp.nyu.edu/~lg691/physComp/w8.html

http://www.birnboim.com/nyu/pcomp/techresearch/shiftregisters.html

http://www.acm.uiuc.edu/sigarch/tutorials/ledarray/

http://www.robotstore.com/store/product.asp?pid=1510&catid=1593

http://www.bgmicro.com/catalog.asp

http://www.armory.com/~spcecdt/electronics/LED_matrix/

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=1406&dDocName=en023805

http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2122

Dead-Ends

http://www.cs.uml.edu/~fredm/courses/91.548-spr03/student/dmeppeli/lab5/

http://www.maxim-ic.com/appnotes.cfm/appnote_number/828

http://www.robotstore.com/store/product.asp?pid=1510&catid=1593

http://www.codemercs.com/IOWarriorE.html

http://www.aselabs.com/articles.php?id=196

http://www.lc-led.com/Catalog/department/35/category/69

http://www.futurlec.com/LEDMatrix.shtml

http://www.geocities.com/SiliconValley/2072/6502prj2.htm

http://www.geocities.com/SiliconValley/2072/eeprom.htm

http://www.phanderson.com/icd/tutorial.html

http://home.iae.nl/users/pouweha/lcd/lcd.shtml

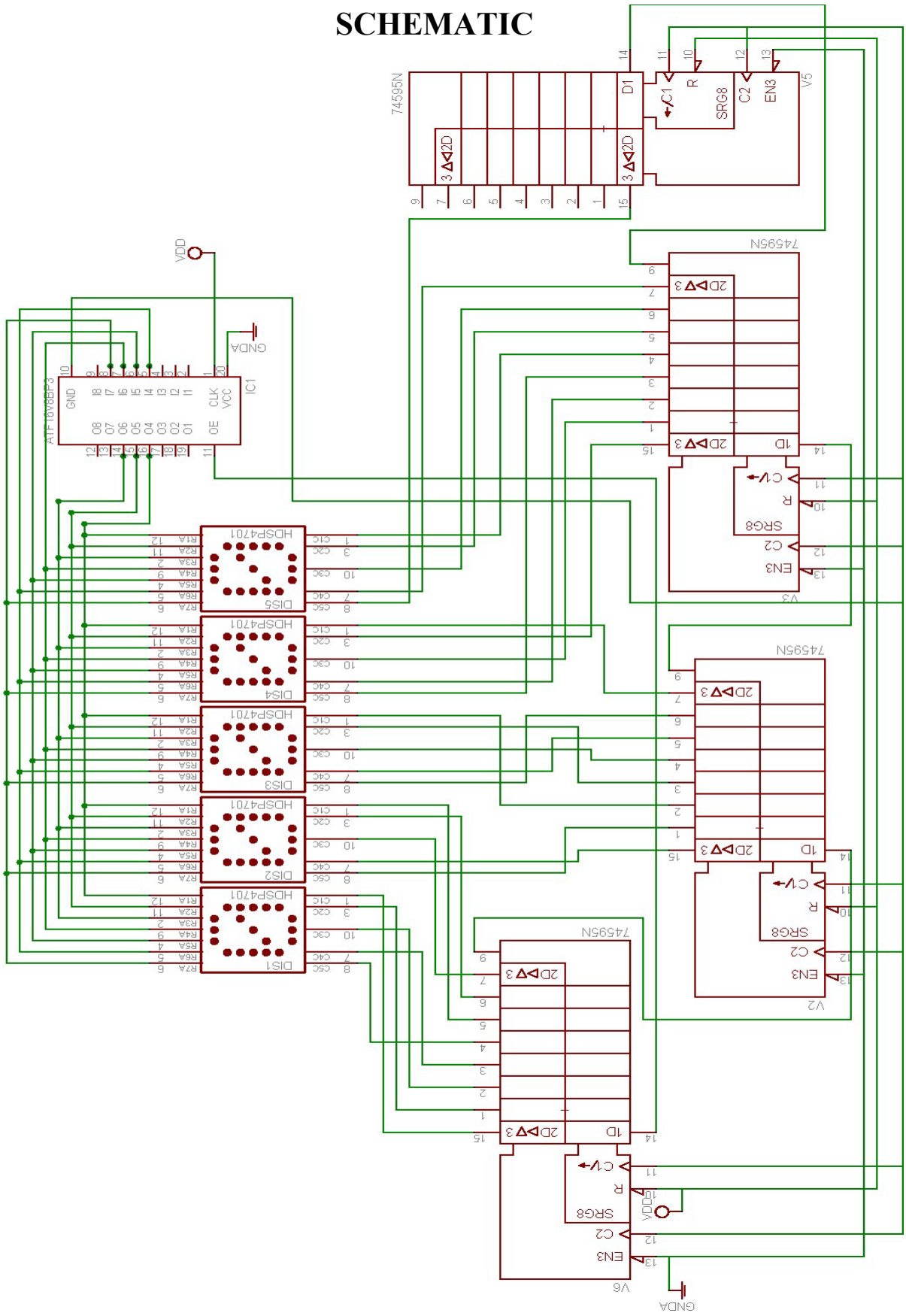
10. Appendix A

CODE

Cddebwfunc.txt	Code for Countdown Decimal Program
HelloJason.txt	Code for 7-Segment Message Program
DMS.txt	Code for Dot Matrix Display Program

10.2 Appendix B

SCHEMATIC



10.3 Appendix C

Programmable LED Marquee

Topic:

My interest is in embedded microprocessors, and practical ways to use them. Nothing is cooler than making things light up or beep, so I want to design a programmable LED Marquee similar to the ones pictured above. I realize that these types of signs are commercially available, and actually building one is most likely beyond the scope of this project due to time and budget constraints, but designing one with the following specifications would use both the logic design and assembly language programming learned in class.

The design specifications are:

The sign can be made to variable lengths by using multiple 5x7 LED matrix modules. (Minimum = 1 module, Maximum = ?? modules).

The sign must be controlled by a microprocessor, which can store a message in memory, and display it on the LED matrix.

This project will consist of several components:

1. **Research:** There are many different ways to make these things, and I will do some Internet research and combine ideas to make my own design.
2. **Design:** Hardware design of the logic circuitry necessary to drive multiple 5x7 LED matrix arrays interfaced to a microprocessor capable of storing data in memory and displaying it on an LED matrix. Most likely it will be patterned after an 8-bit PIC microprocessor, or perhaps a Motorola 68HC microcontroller. (To be determined after research.)
3. **Program**: Write a program in ISA (for whichever microprocessor is chosen) to store the message and to display it.
4. **Implementation:** If there isn't time to actually build it, I hope to be able to at least simulate the hardware using TKGate or the Xilinx FPGA.

Final Paper / Implementation:

The final paper will analyze the research and design process used. It will discuss various possible techniques that could have been implemented, and present the reasons why each was chosen or discarded. I hope to use the design generated by this project to build a working sign sometime in the future when time and budget permit, so real components must be used, and a project plan with cost estimates will be part of the final paper.

10.4 Appendix D

Programmable LED Marquee Project Update

Status:

My initial research started with trying to use an MOS6502 microprocessor and support chips that I had lying around, and simulating it with the FPGA board. In the course of investigation, I learned that I would need to program an EEPROM to hold the data, and I didn't have a clue how to do that. During my research of that, I came across the PicKit2 from Microchip. I had heard of it before, but I had never had the motivation to actually get deep into it until now. I justified the cost ($60) by saying that it was less than a textbook, and I can see myself using it in the future. It comes with a demo board and software for programming the chip with assembly language. After a little more research, I decided to purchase the kit. I also purchased 5 extra chips (they are only about $2 a piece) in case I burned it up and so that I could implement the MCU in future projects.

In my research I also came across a pretty cool chip Maxim, the 6952EA. It is a 5x7 LED dot matrix driver that can support up to 4 LED single color matrixes, or 2 bi-color ones. It also has a built in ASCII 104 character font and memory to store 24 custom fonts. It can be controlled with a microprocessor and a few input lines. At first I balked at the price (about $16), but then I saw a link that said "Order Free Samples". I filled out a request form and they sent me one. Unfortunately, the 2 dot matrix displays I have on hand are Anode-Row, and the Maxim chip drives Cathode-Row displays, so I have to order some displays. I placed an order for some with BG-Micro (about $6) and they will hopefully be in soon. I also ordered some shift registers from Jameco because I found a couple sites that suggested how to use them to drive a row of displays. I also ordered an LCD display with a supposed built-in driver that takes 8-bit ASCII input. I am still waiting for some of the parts.

After the initial research, I spent about a week and half just organizing all my old electronics parts, and assembled a few circuits to re-familiarize myself with the process. Then I spent last weekend going through the initial 12 lessons of the PIC processor and made a working program that I burned into a blank chip and built that on a breadboard.

The next step is to wait for the additional parts to come. While I am waiting I will be figuring out the programming for the PIC and writing up the final paper.

Changes/Deviations:

I don't like to leave anything to chance, so I am approaching this project with a plan A, B, and C.

Plan A is to get the LED displays in time, and assemble the final project as close as possible to the original proposal.

Plan B is to use the LCD display. This will go into effect if Plan A either fails, or gets completed ahead of schedule.

Plan C is to utilize the extra chips in several embedded applications such as a microprocessor controlling a fan by sensing input from a temperature sensor, controlling a stepper motor with limit switches, and controlling or interacting with other simple circuits. Again, Plan C will go into effect in the either the event that A and B fail, or both succeed, and there is extra time.

Timeline:

10/26 Project Proposal

10/30 Initial Internet Research

11/3 Ordered Maxim Chip

11/7 Ordered PicKit2 from Microchip

11/17 Familiarize with Electronics

11/21 Project Update

11/25 Program Pic Chips, Make plan C circuits

12/01 If Chips Arrive: Assemble / Program Display Circuit

12/8 Write Paper.

Preliminary Table of Contents:

10.5 Appendix E

CD-ROM

Photos & Videos

10.6 Appendix F

Datasheets

I. PICkit 2 Starter Kit Manual
II. Low Pin Count Demo Board
III. 8-Bit Shift Register
IV. 7447 BCD to Decimal Decoder
V. LTP-747 5x7 Dot Matrix LED Display
VI. MAN-7 Seven Segment LED Display
VII. PIC 16F690 20 pin 8-bit Flash-Based Microcontroller