A MICROCONTROLLER PROJECT: LED CUBE

PURDUE SCHOOL OF ENGINEERING AND TECHNOLOGY

INDIANA UNIVERSITY PURDUE UNIVERSITY INDIANAPOLIS

INDIANAPOLIS, IN

APRIL 2013

JOHN CURRAN

TABLE OF CONTENTS

Introduction …………………………………………………………………………. 4

Project Requirements ………………………………………………………………... 6

Design Details ……………………………………………………………………….. 7

Hardware Design ……………………………………………………………. 7

Software Design …………………………………………………………….. 8

Implementation Details ……………………………………………………………… 12

Experiment Results ………………………………………………………………...... 16

Discussion …………………………………………………………………………… 16

Conclusion …………………………………………………………………………… 18

References …………………………………………………………………………… 20

Appendices …………………………………………………………………………... 21

PIC18LF4550 Specifications ………………………………………………… 21

NTE823 Specifications……………………………………………………….. 23

Complete C Language Code ………………………………………………….. 25

LIST OF FIGURES

Circuit Schematic ……………………………………………………………………. 7, 15

Component Block Diagram ………………………………………………………….. 10

Operation Sequence Diagram ………………………………………………………... 11

Layout Diagram ………………………………………………………………………. 14

Wiring Diagram ………………………………………………………………………. 15

LIST OF TABLES

List of Parts ………………………………………………………………………….. 12

List of Tools …………………………………………………………………………. 13

INTRODUCTION

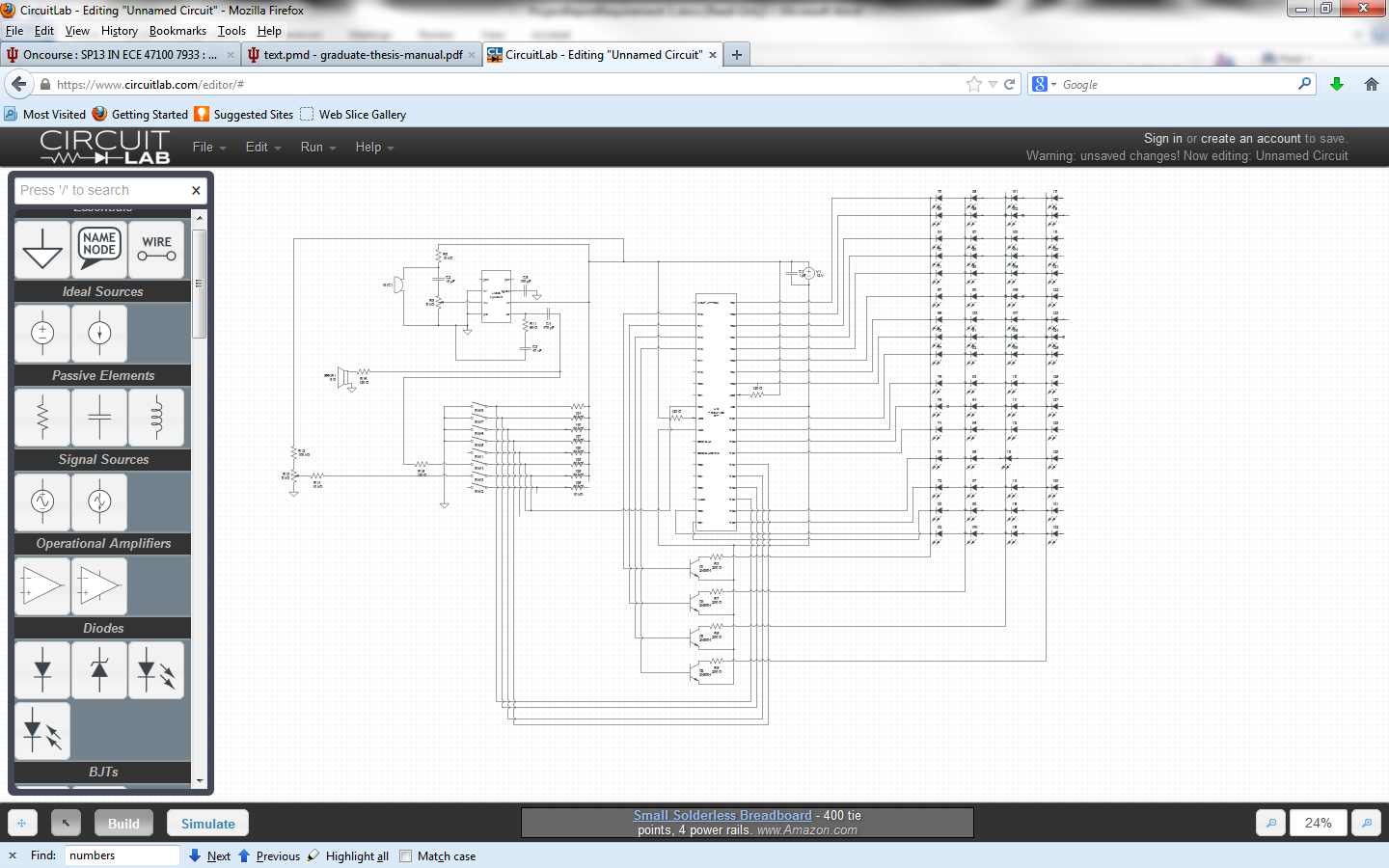
ECE 471 Students are assigned a term project that is required to be an embedded system design. The purpose of the project is for students to gain experience in the development process and cycle of embedded systems. This includes developing a project plan, creating a design, selecting hardware, software development, testing, implementation and documentation. The project is to help students gain confidence in what they have learned throughout their career as engineering students and help them realize how to put all the pieces of knowledge together to create and solve a problem. The project is extremely open ended and students are told to be creative, have fun and do something interesting. There are minimal project requirements that are outlined in a following section. Students will have to present an initial project proposal to Dr. Chien that he must personally approve. The primary consideration that is relevant to the approval of a given project is the size of the project for the number of group members.

The project that I decided to pursue was an LED cube design. The inspiration for this project came from the internet, I have wanted to build a microcontrolled LED cube since I first saw one on Youtube and I knew that ECE 471 would be the perfect opportunity to do this. In this design you control a three dimensional array of LEDs with a minimal number of microcontroller IO pins. You can then create dynamic three dimensional patterns with the LEDs. In some cases, you will not be able to light the LEDs that you need lit at the same time, so you will have to take advantage of what is called persistence of vision. This is the principle that the human eye and brain can only detect certain frequencies of change. This means that if you switch an LED on and off fast enough, the flashing cannot be detected and it appears as if the LED is just on. I decided to pursue a 4x4x4 matrix of LEDs totaling 64. This can easily be done using 64 IO ports, one to power each LED. I powered the same number of LEDs using only 20 IO ports. The anodes of any 4 vertically adjacent LEDs are wired together and all of the cathodes of a 16 LED layer are wired together. This leads to the ability to light any LED in the array by applying voltage to the corresponding of the LED and applying reference ground to the corresponding layer. There are certain further IO pin minimizing techniques (down to 9 IO pins for a 64 LED matrix) and LED connection strategies that take advantage of the unidirectional current flow of diodes, but these limit the design to always having to use multiplexing and persistence of vision to create any pattern because only one LED can be lit at a time. The design I chose allows you to light many LEDs at the same time, but you will still have to use persistence of vision in some cases such as lighting one entire layer and only part of another. Despite the principle of persistence of vision, this leads to better light quality and brighter looking displays because when you switch a large number of LEDs on and off via multiplexing, the duty cycle and average voltage of any single LED drops.

PROJECT REQUIREMENTS

* The LED cube shall use IO.
* The LED cube shall be powered by a microcontroller with no fewer than 24 IO pins and one analog to digital pin.
* The LED cube shall use interrupt.
* The LED cube shall be of 4x4x4 dimensions.
* The LED cube shall use two other peripherals.
  + Peripheral one: timer for real time interrupt
  + Peripheral two: analog to digital converter
* The LED cube shall have selectable patterns via dip switches.
* The LED cube shall have fourteen patterns.
* The LED cube shall use analog to digital conversion to control the speed of the patterns via a potentiometer.
* The LED cube shall have an analog to digital pattern to pulse to music via an electret condenser microphone and amplifier.
* The LED cube shall have a sixteenth “demo” pattern that loops all patterns except the microphone pattern.

DESIGN DETAILS

HARDWARE DESIGN

This is a schematic diagram of the hardware design for the LED cube. The PIC microcontroller is in the center. The LED “layers” are shown in a linear fashion with all of the cathodes connected together and anodes of vertically adjacent LED “columns” of contiguous layers are connected. The cathode “layers” sink, through resistors, to microcontrolled NPN transistors which have their drains connected to ground. The anode “columns” are connected directly to IO ports of the microcontroller. The microphone and amplifier circuit connect to an analog to digital port through a switch. The potentiometer circuit connects to the same analog to digital port through another switch so that these two can be multiplexed into a single port. This can be done because the patterns use either the potentiometer or they use the microphone, never both. The primary reason for this is because I damaged some of the unused analog to digital ports during my experimentation with the amplifier circuit and the potentiometer circuit as this was my first design experience with both of these components. I spent many hours working on the amplifier circuit trying to get everything just right, and with the help of (LM386 Utility Amplifier) I was able to build a functional amplifier circuit. My circuit is not exactly as this source describes, but is an adaptation through experimentation and my needs. The amplifier circuit needed for the analog to digital conversion does not require 200 dB of amplification and this was adapted to only be 20 dB of amplification. There is a large capacitance placed across the power source in order to smooth out the noisy source. This is especially important for the sound amplification circuit so that we obtain as clear a signal as possible from the microphone signal. The amplifier circuit is set up in this state to be able to give a good demonstration to the class via headphones put directly to the microphone, without having to play loud music. However, by simply changing the resistance values and the gain of the amplifier, the circuit can easily be made sensitive enough to respond to ambient sound like

SOFTWARE DESIGN

The software design for the LED cube was simple but somewhat lengthy. There is essentially one function for each pattern along with two delay loop functions and an interrupt service routine function. The program has a state variable which is initialized to zero. The state variable is representative of the state of the dip switches used to control the patterns. The state variable is updated via a real time interrupt, but because this is the only thing done in the interrupt service routine, the current pattern will finish before the selected pattern starts. The real time interrupt is implemented using a timer and fires once a second. Patterns are controlled simply by assigning values three IO ports, two eight bit ports for the sixteen anode columns and one four bit port for the four cathode layers. Patterns are selected conditionally based on the value of the state variable. Because the state variable is initialized to zero, the cube will always start with the same pattern, which is the iterate pattern where each LED is lit individually. This was intentionally done so that the user can see a kind of system check at startup.

Assigning hard coded values to the ports was chosen for three reasons. This is the small size of the cube and the wiring made calculating the needed port values quite easy. The second reason is that since I did not do a fully multiplexed design lighting only one LED at a time, there are far too many permutations of groups of leds that can be lit on the order of . Finally, assigning these hardcoded values directly to the ports is easy due to the relatively small number of patterns. So, for this small cube and the way it is wired it makes more sense just to assign hard coded values to the IO ports. On the other hand, if the cube were larger and utilized multiplexing to only light one LED at a time and or had a far greater number of patterns than 15, it would make much more sense to utilize an array or even a search tree that contains the values for the various LEDs.

There are two different delay loops. One that is hard coded and unchangeable for certain patterns and one that has a variable with a duration dependent upon the digital value obtained form an analog to digital conversion on the potentiometer value.

The second analog to digital function that I have gets the digital representation of the analog signal of the microphone. It then has 16 ranges for the value and outputs an LED representation of this value to the LED cube. In this function, since I need to light entire layers and partial layers at the same time, which is impossible in this design, multiplexing and the principle of persistence of vision are utilized. The cube is divided into 16 lines of LEDs and they grow out and up the cube with increasing values obtained from the analog to digital conversion of the microphone amplifier circuit.

COMPONENT BLOCK DIAGRAM

Delay Duration

Pattern Speed

PATTERN FUNCTIONS

DELAY LOOP

POTENTIOMETER AtoD

IO Ports Set

Pattern Conditional on SV

Discrete Value to Mic Function

PORT-A , PORT-D, PORT-B

MICROPHONE AtoD

STATE VARIABLE

LED CUBE

IO Ports Light Cube

REAL TIME INTERRUPT

Set State Variable

OPERATION SEQUENCE

POWER ON / RESET

RUN CUBE FUNCTION ACCORDING TO CURRENT SV VALUE

READ MIC AtoD VALUE

LOOP FOREVER

DELAY

*IF MIC FUNCTION*

*IF VARIABLE DELAY*

READ POTENTIOMETER AtoD VALUE

1 SECOND INTERVAL

UPDATE STATE VARIABLE

REAL TIME INTERRUPT

INITIALIZE STATE VARIABLE, CONTROLS, AND REGISTERS

RUN ITERATE FUNCTION

IMPLEMENTATION DETAILS

|  |  |  |  |
| --- | --- | --- | --- |
| **Part** | **Price** | **Quantity** | **From** |
| PICkit 3 | 65 | 1 | Microchip.com |
| PIC18LF4550 | 0 | 1 | Microchip.com (sample) |
| 64 5mm LEDs with resistors (Pack of 100) | 11.59 | 1 | Amazon.com |
| 10k ohm audio taper potentiometer | 1.99 | 2 | Fry’s Electronics |
| 1k ohm linear taper potentiometer | 1.99 | 1 | Fry’s Electronics |
| C1815 NPN transistor | 0 | 4 | Previously Owned |
| 16 pin dual inline pin switch | 2.99 | 1 | Fry’s Electronics |
| 10k ohm resistor | 0 | 6 | Previously Owned |
| 3300 uF capacitor | 0 | 2 | Previously Owned Power Supply |
| 100k ohm resistor | 0 | 2 | Previously owned |
| NTE 823 Audio Amp (LM386 equivalent) | 3.99 | 1 | Fry’s Electronics |
| 220 ohm resistor | 0 | 4 | Previously Owned |
| 120 ohm resistor | 0 | 2 | Previously Owned |
| 100 uF capacitor | 0 | 1 | Previously Owned |
| 470 uF capacitor | 0 | 1 | Previously Owned |
| 10 uF capacitor | 0 | 1 | Previously Owned |
| .47 uF capacitor | 0 | 1 | Previously Owned |
| 8 ohm speaker | 0 | 1 | Previously Owned |
| Electret microphone | 0 | 1 | Previously Owned |
| Flick switch | 0 | 1 | Previously Owned |
| Piece of 1x8 lumber | 0 | 1 | Previously Owned |
| Solder | 0 | 0 | Previously Owned |
| 22 gauge solid wire (Spool) | 6.89 | 1 | Fry’s Electronics |
| Zip ties | 0 | 2 | Previously Owned |
| Solderless breadboard | 0 | 1 | Previously Owned |
| Aluminum can | 0 | 1 | Previously Owned |
| Steel craft wire | 0 | 1 | Previously Owned |
| Assorted screws | 0 | 16 | Previously Owned |
|  |  |  |  |
| **Total** | 87.55 | 57 |  |

LIST OF PARTS

LIST OF TOOLS

* PICkit 3 in-circuit debugger
* Wire cutters
* Needle nose pliers
* Wire strippers
* Drill
* Drill bits
* Soldering iron
* Desoldering iron
* Tin snips
* Solderless breadboard
* Screwdriver
* Personal computer
* MPLab X IDE
  + Project Configured to erase all memory before programming chip
  + Power from PICkit3
    - 3.5 Volts
  + Allow PICkit3 to configure memory allocation/map
* C18 compiler
  + Configured for PIC18F4550
* Orbital sander
* Helping hand component holder

Layout Diagram

Smoothing Caps

Wood base

Transistor controlled gnd Array

Microphone

Dip sw

PIC18LF4550

Amp Circuit

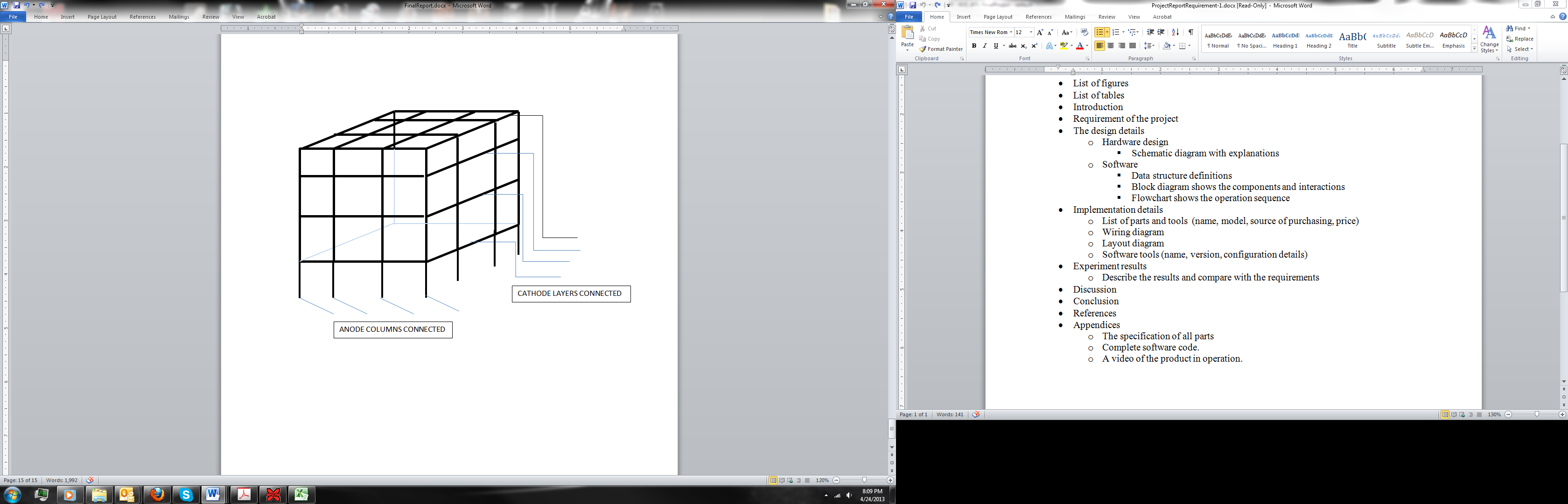
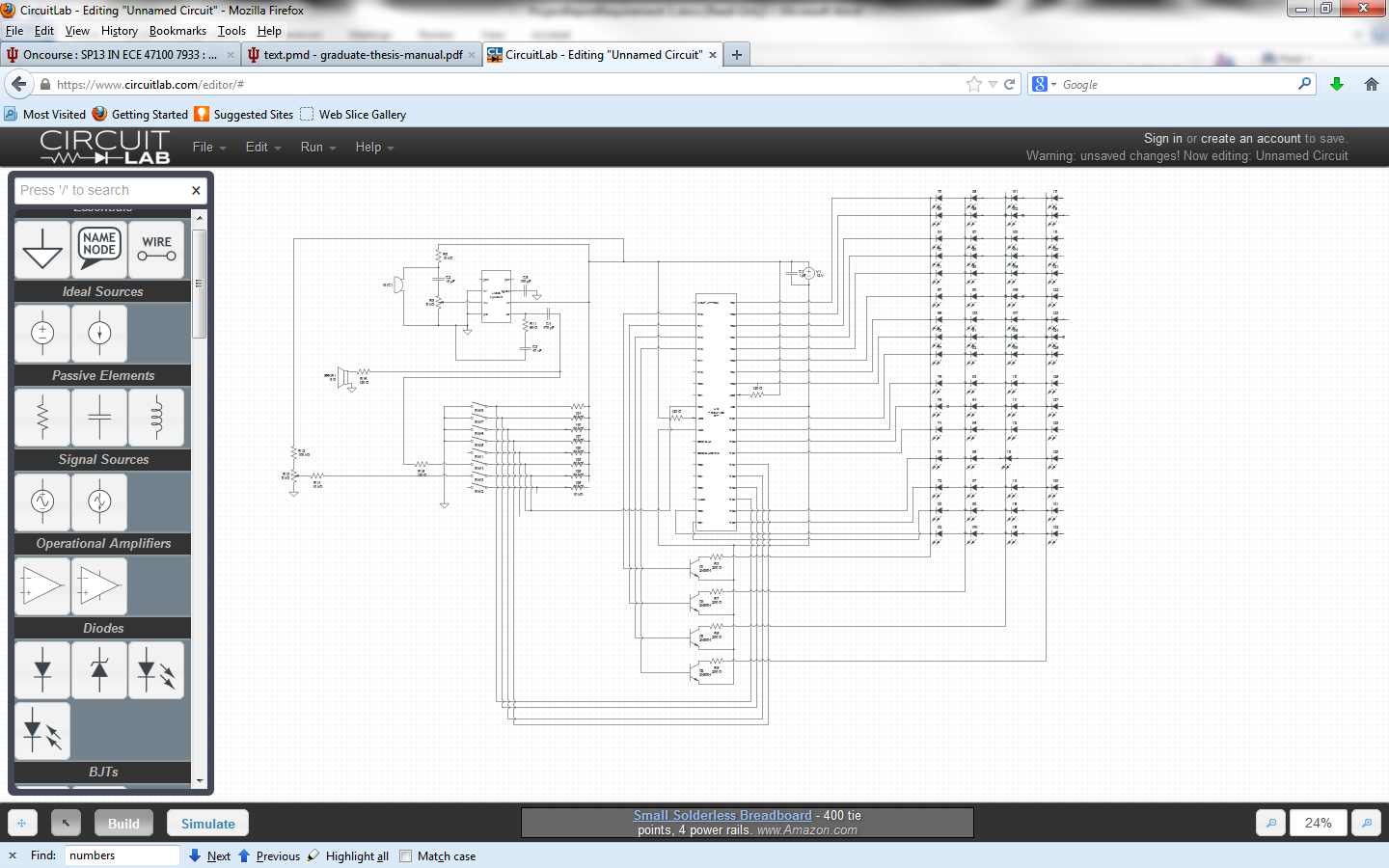
On/Off Switch

Speed Control Potentiometer

Speaker

LED Cube

WIRING DIAGRAM



EXPERIMENT RESULTS

The results of the LED cube project were completely successful. I was able to meet all of the design requirements and specifications. The cube functioned exactly as expected. I Implemented IO and interrupt usage as required. I also used two additional peripherals in the form of analog to digital conversion and timer for a real time interrupt. These met the basic requirements of the ECE 471 project. I was able to implement the number and type of LED cube patterns that I specified in my project proposal. I am able to use switches to select the pattern. I am able to use a microphone to make the cube pulse to music. I was able to implement a pattern speed control, which is an additional complexity that I deemed necessary after my project proposal was approved.

DISCUSSION

ECE 471 and the project therein is something that I have been looking forward to during my undergraduate career at IUPUI in the Purdue School of Engineering and Technology. To me, this class and the ability to complete a project such as this is the culmination of what we have learned as electrical and computer engineering students. This project brings together our knowledge of both software and hardware; we create and solve a problem with that knowledge. Despite my best efforts in my classes so far in my career, this project still proved very challenging. I had a great deal of difficulty using electrical components that I have never implemented before such as an audio amplifier and potentiometer. I have only seen pre-constructed circuits with these components and only done the math for given circuits with known input functions etc. It is a completely new arena when you are trying to design and implement a circuit with unknown input and trying to calculate what resistor, capacitor etc. values that are required. There was a steep learning curve in the electrical components and hardware assembly for me. Looking back, I wish I would have taken ECE 255 as an elective, as it gives a great introduction to the properties of amplifiers, transistors and various other electronic components. Due to my inexperience with these circuits, I exceeded the absolute maximum ratings for voltage and or current on several analog to digital converter channels as several of them stopped working while I was experimenting with the circuitry. Luckily, I was able to correct my mistakes after measuring the inputs at these terminals and determining appropriate resistor values for the circuitry to function within specification. The inspiration for this project came from the internet, I have wanted to build a microcontrolled LED cube since I first saw one on Youtube and I knew that ECE 471 would be the perfect opportunity to do this. I did find, however, that with what we can reasonably be expected to know at this point this project took up significantly more time than I had expected and I ended up dropping a class for the first time due to this time devotion. For this reason, I also feel that the grade weight for this project is inappropriate in comparison with the tests weights because the project does not necessarily have much to do with several of the topics discussed in the class yet can take up most of the time that we have available to devote to one class. This semester has been a little overwhelming with this project taking up far too much time in comparison to the amount that I actually learned doing it. I spent a lot of time trying to figure out small bugs and feel that I did not learn much from fixing them. Looking at the final product of my efforts, it does not seem that my project should have been too difficult to complete in a semesters time, it should have only taken maybe 3 weeks.

CONCLUSION

Despite my setbacks, I still feel that I have learned a lot this semester. The end result of my efforts into this project met every requirement and specification of both Dr. Chien’s open ended project requirements and the requirements that I created for my LED cube implementation. I wish that I could have gone a little bit more into depth than I was able to, but due to my inexperience with the hardware I was not. Had I had or used a little more guidance with the hardware from someone else, I could have had much more time to devote to the complexity and comprehensiveness of my LED cube solution. Improvements that would have been interesting to implement include:

* a true finite state machine software design
* a larger cube, minimizing IO ports using both charlieplexing and or shift registers and or multiplexers
* adding more patterns, using smaller LEDs to prevent shine through
* having multiple input types for audio such as a 3.5 mm stereo plug in addition to the microphone
* having some way to input custom patterns whether that were implemented through control buttons or some communication interface with a PC
* data structures for storing values for various port values and patterns
* having an alarm clock function for the cube

Though I feel that the amount that I learned from my hardware setbacks is disproportionate to the time I devoted to them, I feel that I did learn a lot of other less technical things from this project. First and foremost I learned the importance of the design process and documenting everything that you do so that you have a paper trail and so that you have a reference in case something unexpected happens. Secondly I learned that I should have chosen to work with a partner as having someone with different skills and experience is very beneficial to achieving goals. Finally, I feel that the overall goal that Dr. Chien stated in his project introduction was met; this was that I should gain confidence in working with embedded systems projects. This was my very first time building any electronics project and it was somewhat intimidating, as I knew it would be when I signed up for the class. However, through the completion of this project I realized that I have attained a very good knowledgebase during my time as an engineering student at IUPUI and I do have enough knowledge to overcome problems like this.

# REFERENCES

Alexander, C. K. (2009). Fundamentals of Electric Circuits (Fourth ed.). New York, New York, United States: McGraw-Hill.

LM386 Utility Amplifier. (n.d.). Retrieved March 2013, from web.mit.edu: http://web.mit.edu/6.s28/www/schematics/lm386.htm

Matic, N. (n.d.). PIC microcontrollers, for beginners too. Retrieved February 2013, from groups.csail.mit.edu: http://groups.csail.mit.edu/lbr/stack/pic/pic-microcontrollers.pdf

Matt, S. R. (1998). Electricity and Basic Electronics. The Goodheart-Willcox Company, Inc.

Sendis, C. M. (2006-2009). Understanding the Memory Scheme in the S12(x) Architecture. Retrieved February 2013, from cache.freescale.com: http://cache.freescale.com/files/soft\_dev\_tools/doc/app\_note/AN3784.pdf

PIC18LF4550 datasheet

APPENDICES

PIC18LF4550

**ELECTRICAL CHARACTERISTICS**

**Absolute Maximum Ratings(†)**

Ambient temperature under bias...............................................................................................................-40°C to +85°C

Storage temperature .............................................................................................................................. -65°C to +150°C

Voltage on any pin with respect to VSS (except VDD and MCLR) **(Note 3)** ..................................... -0.3V to (VDD + 0.3V)

Voltage on VDD with respect to VSS ......................................................................................................... -0.3V to +7.5V

Voltage on MCLR with respect to VSS **(Note 2)** ......................................................................................... 0V to +13.25V

Total power dissipation **(Note 1)** ...............................................................................................................................1.0W

Maximum current out of VSS pin ...........................................................................................................................300 mA

Maximum current into VDD pin ..............................................................................................................................250 mA

Input clamp current, IIK (VI < 0 or VI > VDD)20 mA

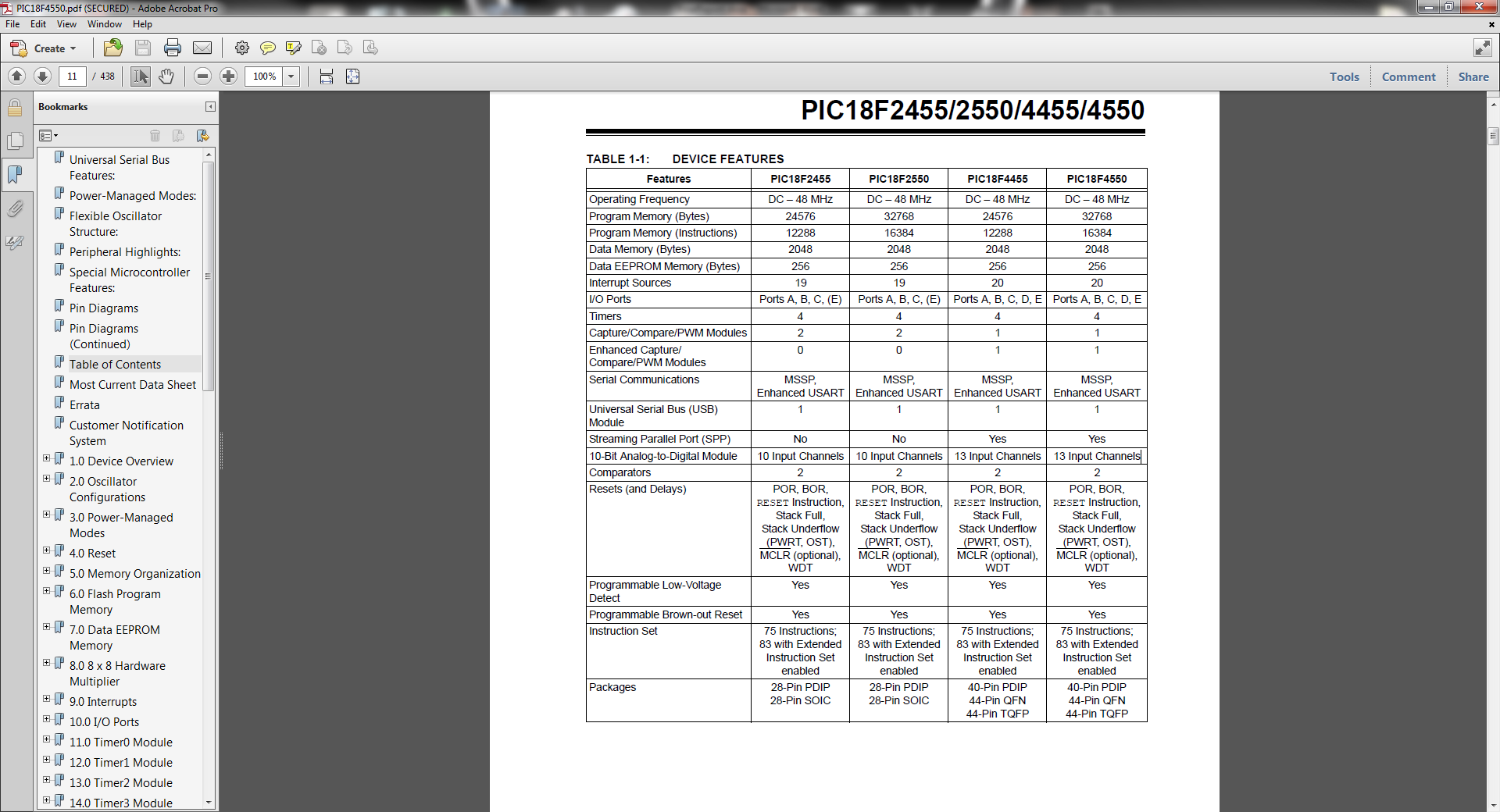
Output clamp current, IOK (VO < 0 or VO > VDD) 20 mA

Maximum output current sunk by any I/O pin..........................................................................................................25 mA

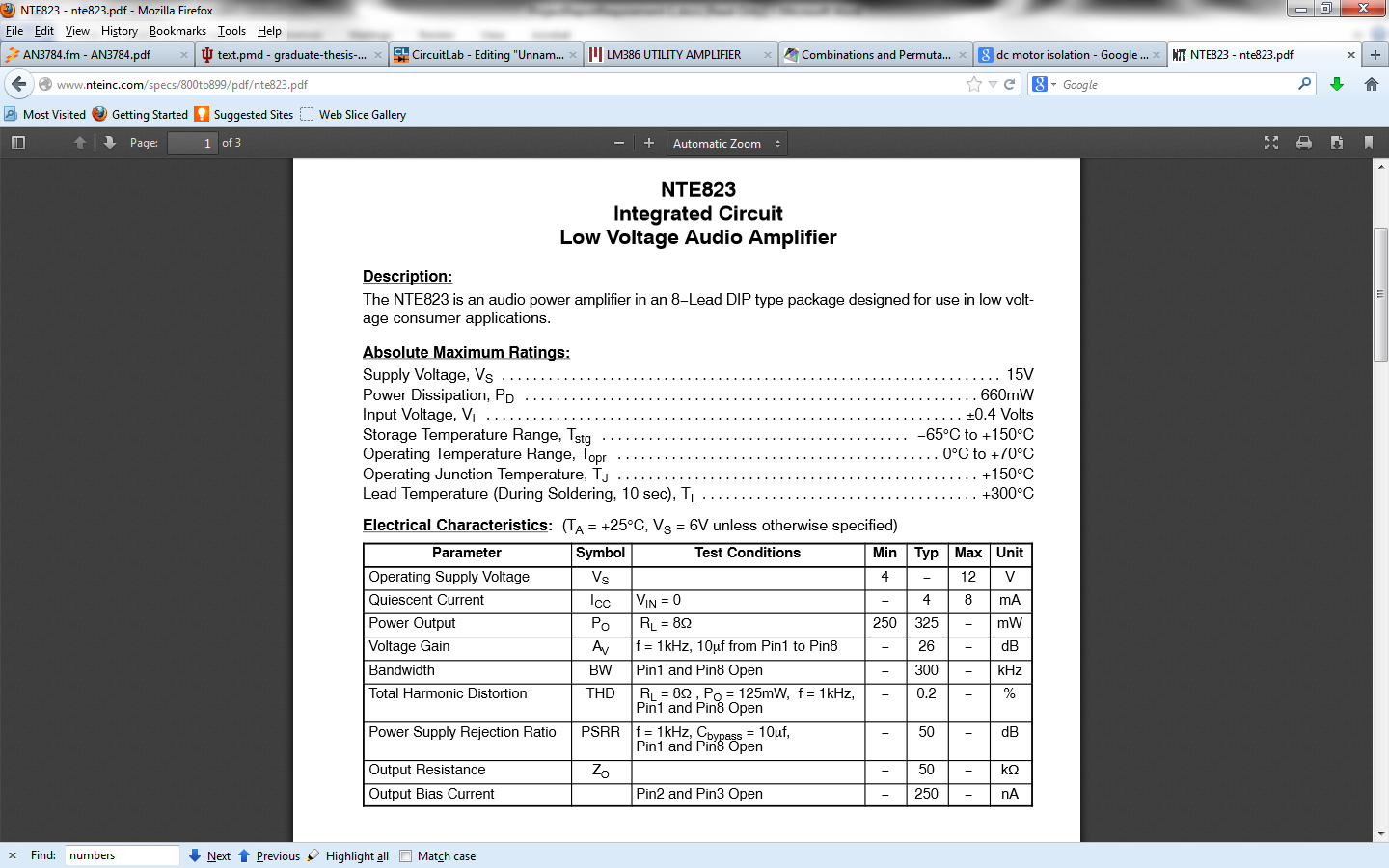
Maximum output current sourced by any I/O pin ....................................................................................................25 mA

Maximum current sunk by all ports .......................................................................................................................200 mA

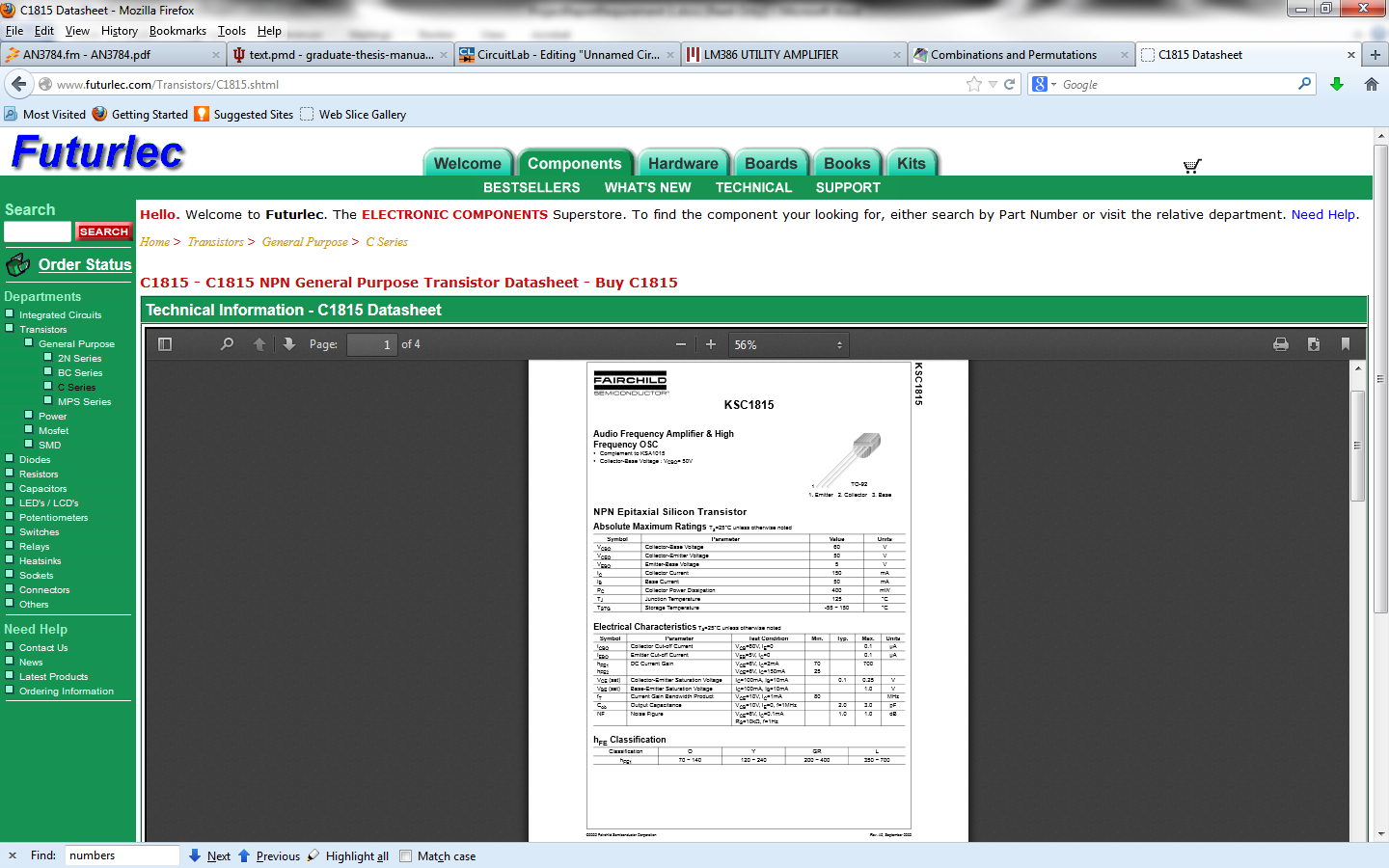
Maximum current sourced by all ports ..................................................................................................................200 Ma



NTE823 (LM386 EQUIVALENT/REPLACEMENT)



C1815



COMPLETE MICROCONTROLLER CODE: C LANGAUGE

#include "p18f4550.h"

#include <p18F4550.h>

//This code was generated automatically using the utility within MPLAB X IDE.

//Window -> PIC memory views -> configuration bits

// CONFIG1L

#pragma config PLLDIV = 1 // PLL Prescaler Selection bits (No prescale (4 MHz oscillator input drives PLL directly))

#pragma config CPUDIV = OSC1\_PLL2// System Clock Postscaler Selection bits ([Primary Oscillator Src: /1][96 MHz PLL Src: /2])

//#pragma config USBDIV = 1 // USB Clock Selection bit (used in Full-Speed USB mode only; UCFG:FSEN = 1) (USB clock source comes directly from the primary oscillator block with no postscale)

// CONFIG1H

#pragma config FOSC = INTOSCIO\_EC// Oscillator Selection bits (Internal oscillator, port function on RA6, EC used by USB (INTIO))

#pragma config FCMEN = OFF // Fail-Safe Clock Monitor Enable bit (Fail-Safe Clock Monitor disabled)

#pragma config IESO = OFF // Internal/External Oscillator Switchover bit (Oscillator Switchover mode disabled)

// CONFIG2L

#pragma config PWRT = OFF // Power-up Timer Enable bit (PWRT disabled)

#pragma config BOR = OFF // Brown-out Reset Enable bits (Brown-out Reset enabled in hardware only (SBOREN is disabled))

//#pragma config BORV = 3 // Brown-out Reset Voltage bits (Minimum setting)

#pragma config VREGEN = ON // USB Voltage Regulator Enable bit (USB voltage regulator disabled)

// CONFIG2H

#pragma config WDT = OFF // Watchdog Timer Enable bit (WDT disabled (control is placed on the SWDTEN bit))

#pragma config WDTPS = 32768 // Watchdog Timer Postscale Select bits (1:32768)

// CONFIG3H

#pragma config CCP2MX = OFF // CCP2 MUX bit (CCP2 input/output is multiplexed with RB3)

#pragma config PBADEN = OFF // PORTB A/D Enable bit (PORTB<4:0> pins are configured as digital I/O on Reset)

#pragma config LPT1OSC = OFF // Low-Power Timer 1 Oscillator Enable bit (Timer1 configured for higher power operation)

#pragma config MCLRE = OFF // MCLR Pin Enable bit (RE3 input pin enabled; MCLR pin disabled)

// CONFIG4L

#pragma config STVREN = OFF // Stack Full/Underflow Reset Enable bit (Stack full/underflow will cause Reset)

#pragma config LVP = OFF // Single-Supply ICSP Enable bit (Single-Supply ICSP disabled)

#pragma config ICPRT = OFF // Dedicated In-Circuit Debug/Programming Port (ICPORT) Enable bit (ICPORT disabled)

#pragma config XINST = OFF // Extended Instruction Set Enable bit (Instruction set extension and Indexed Addressing mode disabled (Legacy mode))

// CONFIG5L

#pragma config CP0 = OFF // Code Protection bit (Block 0 (000800-001FFFh) is not code-protected)

#pragma config CP1 = OFF // Code Protection bit (Block 1 (002000-003FFFh) is not code-protected)

#pragma config CP2 = OFF // Code Protection bit (Block 2 (004000-005FFFh) is not code-protected)

#pragma config CP3 = OFF // Code Protection bit (Block 3 (006000-007FFFh) is not code-protected)

// CONFIG5H

#pragma config CPB = OFF // Boot Block Code Protection bit (Boot block (000000-0007FFh) is not code-protected)

#pragma config CPD = OFF // Data EEPROM Code Protection bit (Data EEPROM is not code-protected)

// CONFIG6L

#pragma config WRT0 = OFF // Write Protection bit (Block 0 (000800-001FFFh) is not write-protected)

#pragma config WRT1 = OFF // Write Protection bit (Block 1 (002000-003FFFh) is not write-protected)

#pragma config WRT2 = OFF // Write Protection bit (Block 2 (004000-005FFFh) is not write-protected)

#pragma config WRT3 = OFF // Write Protection bit (Block 3 (006000-007FFFh) is not write-protected)

// CONFIG6H

#pragma config WRTC = OFF // Configuration Register Write Protection bit (Configuration registers (300000-3000FFh) are not write-protected)

#pragma config WRTB = OFF // Boot Block Write Protection bit (Boot block (000000-0007FFh) is not write-protected)

#pragma config WRTD = OFF // Data EEPROM Write Protection bit (Data EEPROM is not write-protected)

// CONFIG7L

#pragma config EBTR0 = OFF // Table Read Protection bit (Block 0 (000800-001FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR1 = OFF // Table Read Protection bit (Block 1 (002000-003FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR2 = OFF // Table Read Protection bit (Block 2 (004000-005FFFh) is not protected from table reads executed in other blocks)

#pragma config EBTR3 = OFF // Table Read Protection bit (Block 3 (006000-007FFFh) is not protected from table reads executed in other blocks)

// CONFIG7H

#pragma config EBTRB = OFF // Boot Block Table Read Protection bit (Boot block (000000-0007FFh) is not protected from table reads executed in other blocks)

void delay();

void delay3();

void iterate();

void upDownLayer();

void frontBackLayer();

void sideSideLayer();

void frontBackRoll();

void sideSideRoll();

void bouncingCube();

void cubePulse();

void fireworks();

void wholeCube();

void zigzag();

void vLineRoll();

void frontBackLineRoll();

void sideSideLineRoll();

void mic();

void highISR();

int state = 0;

//Timer overflow interrupt service routine to implement real time interrrupt.

//interrupt service routine to update the state variable with the current state of the dip switches

//since this ISR does not call any pattern functions and the pattern fucntion is updated at the end

//of the current pattern, the current pattern will run through in its entirity before the pattern

//function corresponding to the state variable change is called. Interrupt occurs realtively infrequently

//taking into accound the length of time it takes for a switch to be selected and the fact that the current

//pattern runs through its entirity. Between 1-2 seconds for interrupt.

#pragma interrupt highISR

void highISR()

{

state = PORTC;

TMR1H = 0x7F;

TMR1L = 0xFF;

PIR1bits.TMR1IF = 0;

}

#pragma code high\_vector = 0x08

void interrupt\_at\_high()

{

highISR();

}

void main()

{

int i;

state = 0;

//data direction register configuration

TRISD = 0;

TRISB = 0;

TRISA = 0;

TRISE = 0x04;

TRISC = 0xF0;

RCSTA = 0;

TXSTA = 0;

BAUDCONbits.ABDEN = 0;

//usb configuration to be able tu use port c bits as input

UCON = 0x00;

UCFG = 0x08;

UEP1= 0;

//analog to digital configuration

ADCON1 = 0x08;

ADCON2 = 0x88;

ADCON0 = 0x15;

//interrupt configuration

TMR1H = 0x7F;

TMR1L = 0xFF;

RCON &= 0x00;

INTCON = 0xC0;

PIR1 = 0;

IPR1 = 0x01;

T1CON = 0x9D;

PIE1 = 0x01;

//loop forever

while(1)

{

if(state == 0)

iterate();

else if(state == 0x10)

upDownLayer();

else if(state == 0x20)

frontBackLayer();

else if(state == 0x30)

sideSideLayer();

else if(state == 0x40)

frontBackRoll();

else if(state == 0x50)

sideSideRoll();

else if(state == 0x60)

cubePulse();

else if(state == 0x70)

bouncingCube();

else if(state == 0x80)

fireworks();

else if(state == 0x90)

zigzag();

else if(state == 0xA0)

vLineRoll();

else if(state == 0xB0)

frontBackLineRoll();

else if(state == 0xC0)

sideSideLineRoll();

else if(state == 0xD0)

wholeCube();

else if(state == 0xE0)

mic();

else if(state == 0xF0)

{

iterate();

for(i = 0; i < 6; i++)

upDownLayer();

for(i = 0; i < 6; i++)

frontBackLayer();

for(i = 0; i < 6; i++)

sideSideLayer();

for(i = 0; i < 6; i++)

frontBackRoll();

for(i = 0; i < 6; i++)

sideSideRoll();

for(i = 0; i < 6; i++)

bouncingCube();

for(i = 0; i < 6; i++)

cubePulse();

for(i = 0; i < 6; i++)

fireworks();

for(i = 0; i < 6; i++)

zigzag();

for(i = 0; i < 6; i++)

vLineRoll();

for(i = 0; i < 6; i++)

frontBackLineRoll();

for(i = 0; i < 6; i++)

sideSideLineRoll();

for(i = 0; i < 20000; i++)

wholeCube();

}

}

}

//this fucntion will light every LED sequentially

void iterate()

{

PORTB = 0;

PORTD = 0;

PORTA = 0x01;

PORTD = 0x01;

delay();

PORTD = 0x02;

delay();

PORTD = 0x04;

delay();

PORTD = 0x08;

delay();

PORTD = 0x10;

delay();

PORTD = 0x20;

delay();

PORTD = 0x40;

delay();

PORTD = 0x80;

delay();

PORTD = 0;

PORTB = 0x01;

delay();

PORTB = 0x02;

delay();

PORTB = 0x04;

delay();

PORTB = 0x08;

delay();

PORTB = 0x10;

delay();

PORTB = 0x20;

delay();

PORTB = 0x40;

delay();

PORTB = 0x80;

delay();

PORTB = 0;

PORTA = 0x02;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PORTD = 0x01;

delay();

PORTD = 0x02;

delay();

PORTD = 0x04;

delay();

PORTD = 0x08;

delay();

PORTD = 0x10;

delay();

PORTD = 0x20;

delay();

PORTD = 0x40;

delay();

PORTD = 0x80;

delay();

PORTD = 0;

PORTB = 0x01;

delay();

PORTB = 0x02;

delay();

PORTB = 0x04;

delay();

PORTB = 0x08;

delay();

PORTB = 0x10;

delay();

PORTB = 0x20;

delay();

PORTB = 0x40;

delay();

PORTB = 0x80;

delay();

PORTB = 0;

PORTA = 0x04;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PORTD = 0x01;

delay();

PORTD = 0x02;

delay();

PORTD = 0x04;

delay();

PORTD = 0x08;

delay();

PORTD = 0x10;

delay();

PORTD = 0x20;

delay();

PORTD = 0x40;

delay();

PORTD = 0x80;

delay();

PORTD = 0;

PORTB = 0x01;

delay();

PORTB = 0x02;

delay();

PORTB = 0x04;

delay();

PORTB = 0x08;

delay();

PORTB = 0x10;

delay();

PORTB = 0x20;

delay();

PORTB = 0x40;

delay();

PORTB = 0x80;

delay();

PORTB = 0;

PORTA = 0x08;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PORTD = 0x01;

delay();

PORTD = 0x02;

delay();

PORTD = 0x04;

delay();

PORTD = 0x08;

delay();

PORTD = 0x10;

delay();

PORTD = 0x20;

delay();

PORTD = 0x40;

delay();

PORTD = 0x80;

delay();

PORTD = 0;

PORTB = 0x01;

delay();

PORTB = 0x02;

delay();

PORTB = 0x04;

delay();

PORTB = 0x08;

delay();

PORTB = 0x10;

delay();

PORTB = 0x20;

delay();

PORTB = 0x40;

delay();

PORTB = 0x80;

delay();

PORTB = 0;

}

//This LED will light the layers of the cube in ascending and then decending order

void upDownLayer()

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

delay();

PORTA = 0x02;

delay();

PORTA = 0x04;

delay();

PORTA = 0x08;

delay();

PORTA = 0x04;

delay();

PORTA = 0x02;

delay();

}

//This cube will light the vertical cube layers sequentially from front to back

void frontBackLayer()

{

PORTA = 0x0F;

PORTD = 0;

PORTB = 0;

PORTD = 0x0F;

delay();

PORTD = 0xF0;

delay();

PORTD = 0;

PORTB = 0x0F;

delay();

PORTB = 0xF0;

delay();

PORTB = 0x0F;

delay();

PORTB = 0;

PORTD = 0xF0;

delay();

}

//this function will light the vertical layers sequentially from side to side

void sideSideLayer()

{

PORTA = 0x0F;

PORTD = 0x11;

PORTB = 0x11;

delay();

PORTD = 0x22;

PORTB = 0x22;

delay();

PORTD = 0x44;

PORTB = 0x44;

delay();

PORTD = 0x88;

PORTB = 0x88;

delay();

PORTD = 0x44;

PORTB = 0x44;

delay();

PORTD = 0x22;

PORTB = 0x22;

delay();

}

//This function will light the outer layers of the cube in a front to back rolling pattern

void frontBackRoll()

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

delay();

PORTA = 0x0F;

PORTB = 0;

PORTD = 0x0F;

delay();

PORTA = 0x08;

PORTD = 0xFF;

PORTB = 0xFF;

delay();

PORTA = 0x0F;

PORTD = 0;

PORTB = 0xF0;

delay();

}

//this function will light the outer layers of the cube in a side to side rolling pattern

void sideSideRoll()

{

PORTA = 0x0F;

PORTD = 0x11;

PORTB = 0x11;

delay();

PORTA = 0x08;

PORTD = 0xFF;

PORTB = 0xFF;

delay();

PORTD = 0x88;

PORTB = 0x88;

PORTA = 0x0F;

delay();

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

delay();

}

//this function will light a 2x2x2 cube at the center and then delay and then light the whole 4x4x4 cube

void cubePulse()

{

int j = 0;

PORTD = 0x60;

PORTB = 0x06;

PORTA = 0x06;

delay();

PORTD = 0;

PORTB = 0;

PORTA = 0;

delay();

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x0F;

delay();

PORTD = 0;

PORTB = 0;

PORTA = 0;

delay();

}

//this function will light a 2x2x2 cube and move it through the larger cube in a bouncing diagonal pattern

void bouncingCube()

{

PORTA = 0x03;

PORTD = 0x30;

PORTB = 0x03;

delay();

PORTA = 0x06;

PORTB = 0;

PORTD = 0x66;

delay();

PORTA = 0x0C;

PORTD = 0xC0;

PORTB = 0x0C;

delay();

PORTA = 0x06;

PORTD = 0;

PORTB = 0x66;

delay();

}

//This function will light a central LED in a linear ascending pattern like an ascending

//fireworks shell and then it will light the entire top layer like the firework bursting

//then it will have sparse led spots trickle down from top to bottom of the cube

//to simulate the sparks falling

void fireworks()

{

PORTA = 0x01;

PORTB = 0;

PORTD = 0x40;

delay3();

PORTA = 0x02;

delay3();

PORTA = 0x04;

delay3();

PORTA = 0x08;

delay3();

PORTD = 0xFF;

PORTB = 0xFF;

delay3();

PORTD = 0xA6;

PORTB = 0x3C;

PORTA = 0x0C;

delay3();

PORTD = 0x6A;

PORTB = 0xC3;

PORTA = 0x06;

delay3();

PORTD = 0x48;

PORTB = 0x39;

PORTA = 0x03;

delay3();

PORTA = 0x01;

PORTB = 0x02;

PORTD = 0;

delay3();

PORTA = 0x02;

delay();

PORTA = 0x04;

delay3();

PORTA = 0x08;

delay3();

PORTD = 0xFF;

PORTB = 0xFF;

delay3();

PORTD = 0xA6;

PORTB = 0x3C;

PORTA = 0x0C;

delay3();

PORTD = 0x6A;

PORTB = 0xC3;

PORTA = 0x06;

delay3();

PORTD = 0x39;

PORTB = 0x48;

PORTA = 0x03;

delay3();

}

//This function will light a single led on an outer layer and then move to another LED

//in a vertically diagonal direction until a top or bottom corner is reached and then

//the program moves the lit LED in the opposite vertical direction along the nextouter layer

void zigzag()

{

PORTA = 0x01;

PORTD = 0x01;

PORTB = 0;

delay();

PORTA = 0x02;

PORTD = 0x02;

delay();

PORTA = 0x04;

PORTD = 0x04;

delay();

PORTA = 0x08;

PORTD = 0x08;

delay();

PORTA = 0x04;

PORTD = 0x80;

delay();

PORTA = 0x02;

PORTD = 0;

PORTB = 0x08;

delay();

PORTA = 0x01;

PORTB = 0x80;

delay();

PORTA = 0x02;

PORTB = 0x40;

delay();

PORTA = 0x04;

PORTB = 0x20;

delay();

PORTA = 0x08;

PORTB = 0x10;

delay();

PORTA = 0x04;

PORTB = 0x01;

delay();

PORTA = 0x02;

PORTD = 0x10;

PORTB = 0;

delay();

}

//This function simply lights the whole cube

void wholeCube()

{

PORTA = 0x0F;

PORTD = 0xFF;

PORTB = 0xFF;

}

//This function lights and moves a vertical line of LEDS in a circuilar pattern around the outer layers of the cube

void vLineRoll()

{

PORTD = 0x01;

PORTB = 0;

PORTA = 0x0F;

delay();

PORTD = 0x02;

delay();

PORTD = 0x04;

delay();

PORTD = 0x08;

delay();

PORTD = 0x80;

delay();

PORTD = 0;

PORTB = 0x08;

delay();

PORTB = 0x80;

delay();

PORTB = 0x40;

delay();

PORTB = 0x20;

delay();

PORTB = 0x10;

delay();

PORTB = 0x01;

delay();

PORTD = 0x10;

PORTB = 0;

delay();

}

//This function does the same as the vertical line roll but with a horizontal line role in a circuilar

//direction around the outer layers of the cube

void frontBackLineRoll()

{

PORTD = 0x0F;

PORTB = 0x00;

PORTA = 0x01;

delay();

PORTA = 0x02;

delay();

PORTA = 0x04;

delay();

PORTA = 0x08;

delay();

PORTD = 0xF0;

delay();

PORTD = 0;

PORTB = 0x0F;

delay();

PORTB = 0xF0;

delay();

PORTA = 0x04;

delay();

PORTA = 0x02;

delay();

PORTA = 0x01;

delay();

PORTB = 0x0F;

delay();

PORTB = 0;

PORTD = 0xF0;

delay();

}

//This function is the same as the previous frontBackLineRoll but moves the horizontal line in the

//other direction, meaning this one rolls from side to side where as the other rolled from front to back

void sideSideLineRoll()

{

PORTD = 0x11;

PORTB = 0x11;

PORTA = 0x01;

delay();

PORTA = 0x02;

delay();

PORTA = 0x04;

delay();

PORTA = 0x08;

delay();

PORTD = 0x22;

PORTB = 0x22;

delay();

PORTD = 0x44;

PORTB = 0x44;

delay();

PORTD = 0x88;

PORTB = 0x88;

delay();

PORTA = 0x04;

delay();

PORTA = 0x02;

delay();

PORTA = 0x01;

delay();

PORTD = 0x44;

PORTB = 0x44;

delay();

PORTD = 0x22;

PORTB = 0x22;

delay();

}

//This function reads a value from the analog to digital converter in the range of 0-1023 and then

//ranges this number in 16 evenly divided portions of the 1024 value range and then outputs a corresponding

//level of LEDs to light in response to the value. The higher the value read, the more leds to light.

//It lights sequentially more horizontal lines of leds, 16 lines in total for the highest value.

//It utilizes a fixed loop to "hold" the value so the lighting of the LEDs is slow enough to be visible

//but not so slow that it misses reasonably fast beats

void mic()

{

int i;

int adTemp = 0;

ADCON0 = 0x1D;

;

ADCON0 |= 0x02;

while(ADCON0bits.GO\_NOT\_DONE != 0)

;

adTemp = ADRES;

ADCON0bits.ADON = 0;

i = 20;

PORTA = 0;

PORTB = 0;

PORTD = 0;

while(i--)

{

if(adTemp <= 63)

{

PORTA = 0x01;

PORTD = 0x11;

PORTB = 0x11;

}

else if( (adTemp <= 127))

{

PORTA = 0x01;

PORTD = 0x33;

PORTB = 0x33;

}

else if((adTemp <= 191))

{

PORTA = 0x01;

PORTD = 0x77;

PORTB = 0x77;

}

else if(adTemp <= 255)

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

}

else if((adTemp <= 319))

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x02;

PORTB = 0x11;

PORTD = 0x11;

}

else if((adTemp <= 383))

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x02;

PORTB = 0x33;

PORTD = 0x33;

}

else if(adTemp <= 447)

{

PORTA = 0x01;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x02;

PORTB = 0x77;

PORTD = 0x77;

}

else if((adTemp <= 511))

{

PORTA = 0x03;

PORTD = 0xFF;

PORTB = 0xFF;

}

else if((adTemp <= 575))

{

PORTA = 0x03;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x04;

PORTB = 0x11;

PORTD = 0x11;

}

else if((adTemp <= 639))

{

PORTA = 0x03;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x04;

PORTB = 0x33;

PORTD = 0x33;

}

else if(adTemp <= 703)

{

PORTA = 0x03;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x04;

PORTB = 0x77;

PORTD = 0x77;

}

else if((adTemp <= 767))

{

PORTA = 0x07;

PORTD = 0xFF;

PORTB = 0xFF;

}

else if((adTemp <= 831))

{

PORTA = 0x07;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x08;

PORTB = 0x11;

PORTD = 0x11;

}

else if((adTemp <= 895))

{

PORTA = 0x07;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x08;

PORTB = 0x33;

PORTD = 0x33;

}

else if((adTemp <= 959))

{

PORTA = 0x07;

PORTD = 0xFF;

PORTB = 0xFF;

PORTA = 0x08;

PORTB = 0x77;

PORTD = 0x77;

}

else

{

PORTA = 0x0F;

PORTD = 0xFF;

PORTB = 0xFF;

}

}

}

//This function reads a value from the A to D converter which will be hooked to the potentiometer

//and then converts this value in the range from 0-1023 into a counter for a do othing while loop

//creating a variable delay loop. This controls the speed of the LED cube patterns

void delay()

{

int i;

ADCON0 = 0x1D;

;

ADCON0 |=2;

while(ADCON0bits.GO\_NOT\_DONE != 0)

;

i = ADRES\*20 + 100;

ADCON0bits.ADON = 0;

while(i)

i--;

}

//This is a fixed value delay loop that is perfect for the fireworks pattern, I see no reason

//to have the fireworks pattern on a variable speed as it detracts from the illusion instead of

//adding to it.

void delay3()

{

int i = 5000;

while(i)

i--;

}