THE TRICOLOR LED VISULIZER

EMBEDDED SYSTEMS FINAL PROJECT REPORT

ECE310-01 EMBEDDED SYSTEMS

16 November 2005

Team ABJC
"Hear No Evil, Speak No Evil"

Andrew Boice
Jin Chen

Table of Contents

Introduction

We live in a day and age where our modern economy thrives on innovative ideas, even if the value of those ideas is not immediately apparent. Some of the most successful products have sprung from what are seemingly mundane contrivances. Our primary goal with this embedded project was to create something innovative that does not, at least does not to our knowledge, already exist. After some brainstorming, we decided on a project that would be both interesting to design and entertaining to use once the design phase was complete. We wanted to create something that would be both visual and interactive. The idea that we choose from the result of our brainstorming was a "rotating multi-color LED VU meter." The device will be a column of tri-color LEDs rotating at high-speed that display an output that is indicative of the level of sound in the general vicinity. It will use a microphone to detect the level of sound and then a microcontroller to handle displaying a corresponding pattern on the spinning LEDs.

Objectives and Specifications

Our main objective was to complete a LED VU meter that responds to ambient sound in its general vicinity and induces visual illusion of circular rings of light when rotating at high speed. These rings of light must be able to switch between colors of red, green, and blue without visual irregularities, and preferable should rotate at or higher than 1800 RPM (30 frames per second) in order to induce the visual illusion. A generally accepted VU meter theme or lighting sequence of the LEDs should be adapted, and two independent power sources must be implemented, one for the stationary base motor, and one for the rotating cylinder. The device should be easily reprogrammable and upgradeable. Circuitry and controller should be encapsulated and hidden from view.

Currently, we have successfully achieved all required specifications described above. The cylinder is rotates at 3600 RPM at max speed and induces the desired visual illusion. A VU meter theme where blue LEDs are on when no sound is detected, green LEDs are on when sound is detected, and a single red LED used represent the max level of sound detected has been implemented. For simplicity an AC source was selected to power the base rotational motor, and 4 AA batteries server as the power source for the LEDs and controller circuitry. All circuitry, except the LEDs and the power switches and microphone, are encapsulated within the cylinder and hidden out of view.

Strategy and Implementation

A Freescale Microcontroller module containing an MC9S12C32, one electret microphone, microphone amplifying circuitry, (4) LED driver chips, and 20 tri-color LEDs were used to construct our device (see Hardware Schematic in Appendix A).

The Motorola microcontroller chip MC9S12C32 is the central control unit of our device. This unit measures the analog output of the microphone and converts the data into digital values which are used to determine the on and off status of each 20 tri-color LEDs. The (4) LED drivers are connected to the microcontroller via the SPI interface which allows the microcontroller to enable or disable each LED.

Analog to Digital (ATD) port 0 is used to convert the analog output from the microphone to an 8 bit resolution digital value. The analog output of the microphone is a continuous analog signal with amplitude of approximately 30mV. This output contains both positive and negative peaks with respect to the system ground. In order to utilize both the negative and positive portion of this signal without hardware rectification, the microphone signal is amplified and offset by a DC value of +2.5 V. This allows the sound signal detected to stay within the whole 0 to 5 V region. The converted data collected from the ATD port is then digitally rectified through a simple algorithm with 2.5 V as the minimum input and 0 and 5 V as maximums. Consequentially, our 8 bit resolution is effectively reduce to 7 bits, however, since only 20 LEDs are being controlled, 7 bit resolution (0 to 128) is quite sufficient.

The 20 tri-color LEDs are actually consisted of 60 single colored LEDs, therefore, 4 LED controllers, each capable of controlling 16 LEDS (64 total capability), are implemented to control the 20 tri-color LEDs. These 4 LED drivers use simple 16 bit shift registers which latch into an output register to set the state of each attached LED. They also provide constant current drive of the LEDs thus eliminating the need for current limiting resistors. The drivers receive their inputs from the SPI port (PTM0) of the micro-controller, which clocks sequences of 8 data bytes to fill all the divers out of port PTM0, and latches the shifted in values for all (4) driver chips simultaneously via a pulse on PTM3.

Firmware code on the MC9S12C32 initiates the microcontroller's modules and governs the sequence and timing of the data sent out of the SPI and latch port.

Since we are not as familiar to the field of mechanical mechanisms as we are to electrical systems, the mechanical portion of our project was conducted through a mixture of trial and error and advice from more knowledgeable sources.

Testing Procedure and Results

Software:
Stages of the software were defined according to the functionality of each set of code. These stages included the following:

- Micro-controller initialization
    - ATD initialization
    - SPI initialization
    - Port PTM3 initialization
- ATD conversion
- SPI sending procedure
- LED theme implementation
    - Determine LED location
    - Differentiate between LED colors
    - Synchronize LED lighting sequence with ATD signal received
    - Adjust the sensitivity of the LEDs
- Run the firmware on the controller independent from the simulating environment.

Each of these stages was tested separately in sequential order

Hardware:

- Design and implementation the base that houses the motor
- Secure motor into the base housing
- Attach the cylinder on to the motor shaft
- Install the circuitry with only the switches and LEDs visible
- Balance the cylinder to maximize rotating speed.

Each of these stages was tested separately in sequential order.

Bill of Materials

The following is the approximate costs of the materials used to construct our materials.

Material	**Description**	**Quantity**	**Cost**
Motor	AC, 2000RPM (from fan)	1	$15.00
Microphone	0.38 inch Diameter Electret	2	$1.00
Op Amps	Signal Rail Op-Amp	1	$0.50
Battery Holder	4AA - 2.5 x 1 inch	1	$1.60
LED Driver Chip	1 chip can drive 16 LEDs	4	$10.00
Tricolor LED	RGB 4-Lead Tri-Color LED	20	$8.00
PVC Pipe	Cardboard Tube – 2" Diameter	1	In Kind
Optical Sensor	For timing detection	1	In Kind
Micro Controller Kit	Given in class	1	$25.00
PCB	Made by Rose	1	In Kind
Total			**49.35**

User Manual

To Turn On:

1. Set the device on a soft surface that absorbs vibration well.
2. Plug in the motor to 120V AC outlet
3. Turn on the LEDs by a switch located at the top of the device, the 20 LEDs should all light up.
4. Turn on the motor by pushing the red push button located at the side of the base housing.
5. Crank up the music and be amazed.

To Turn Off:

1. Turn off the motor by pushing the red push button located at the side of the base housing.
2. Wait for device to stop spinning.
3. Turn off the LEDs by flipping the switch located at the top of the device - all 20 LEDs should go out.

NOTE: Do not touch LED column while motor is still spinning as this might prove damaging to both user and device

Conclusion and Future Plans

Both members of the team thoroughly enjoyed this project, and seeing our idea materialize and working is the greatest reward. This project allowed us to achieve a higher understanding of the embedded process. However, due to time constraints, we were not able to reach the full potential of our design. Many enhancements can still be made to make this project even better. It is our hope that in the future many more of these potential applications can be explored and new feature implemented. Research on the marketability of this product will also be researched and pursued.

References and Acknowledgements

Our source information on programming the MC9S12C32 came from the Reference Manual of the HCS12 Micro-Controllers from Freescale Semiconductors, Inc. provided to us by our professor Dr. Jian Jian Song.

We would like to sincerely thank the following individuals for their help in making this project a reality.

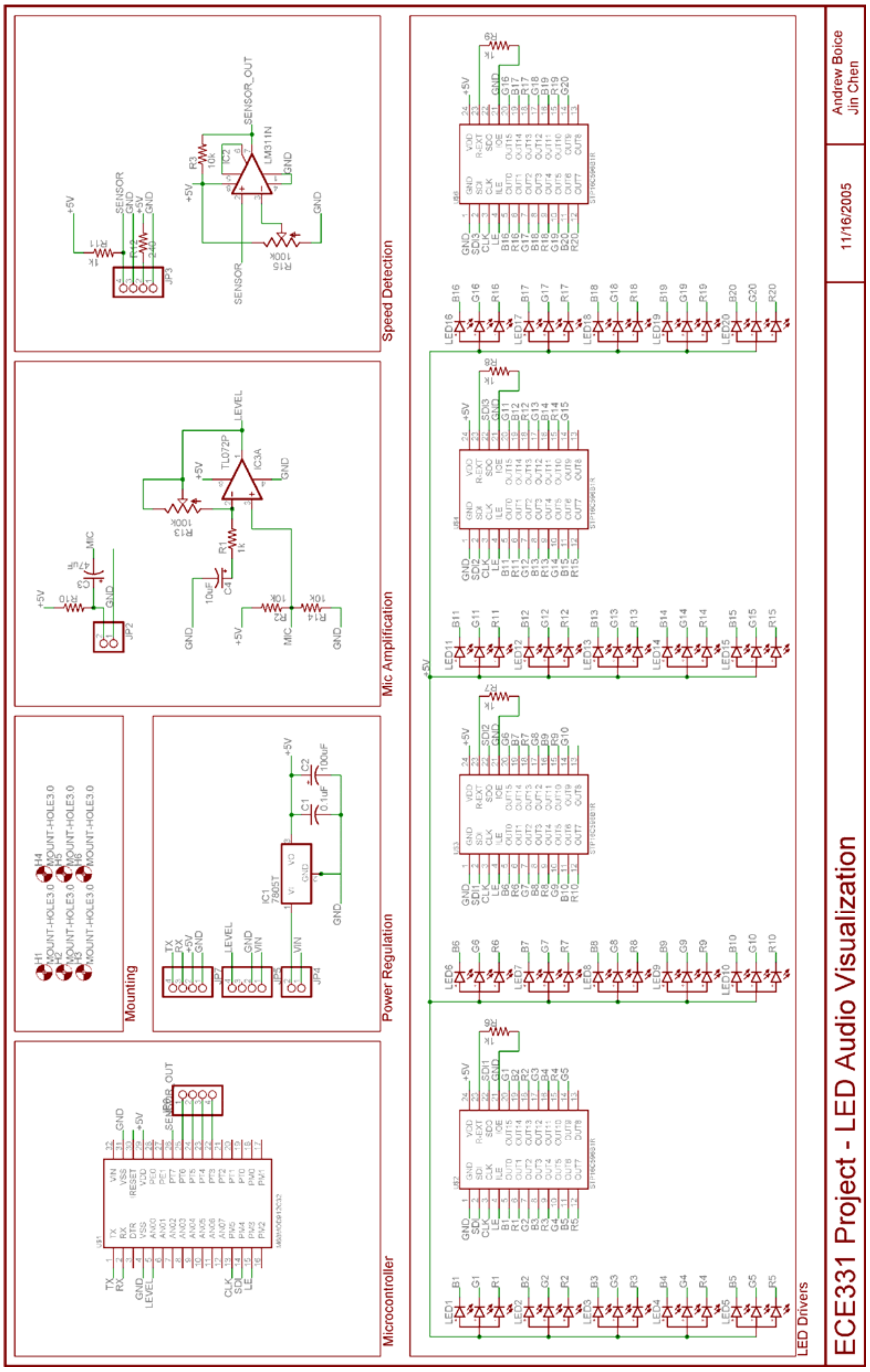
Dr. Jian Jian Song
For his advice and parts (light sensor and DIP sockets)

David T Mumaw
For his help in some of the mechanical aspects of our project. He voluntarily offered to custom machined an aluminum shaft mount for the light column.

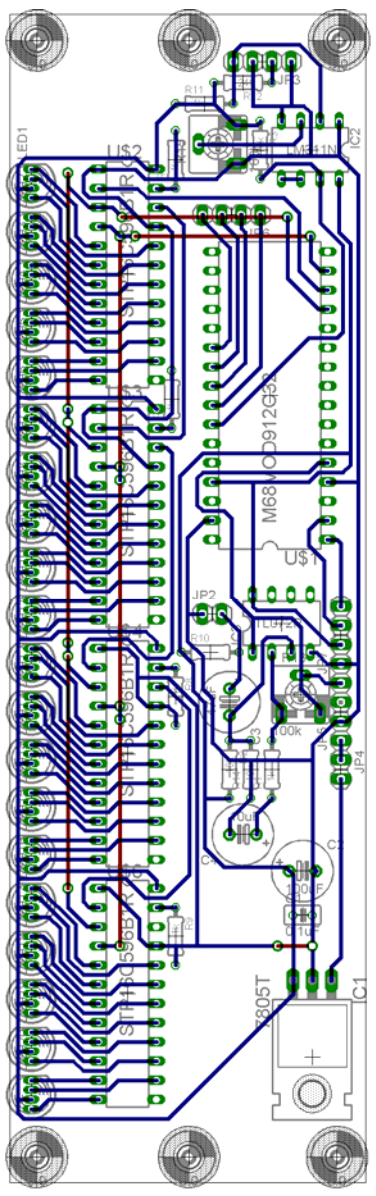
Gary D Burgess
For providing us access to the PCB fabrication machinery and allowing us to use a spare cardboard tube for as our column.

Greg Stump
For allowing us to use the Hatfield Hall scene shop woodworking tools and assorted scrap wood to construct the motor mount base and machine the column.

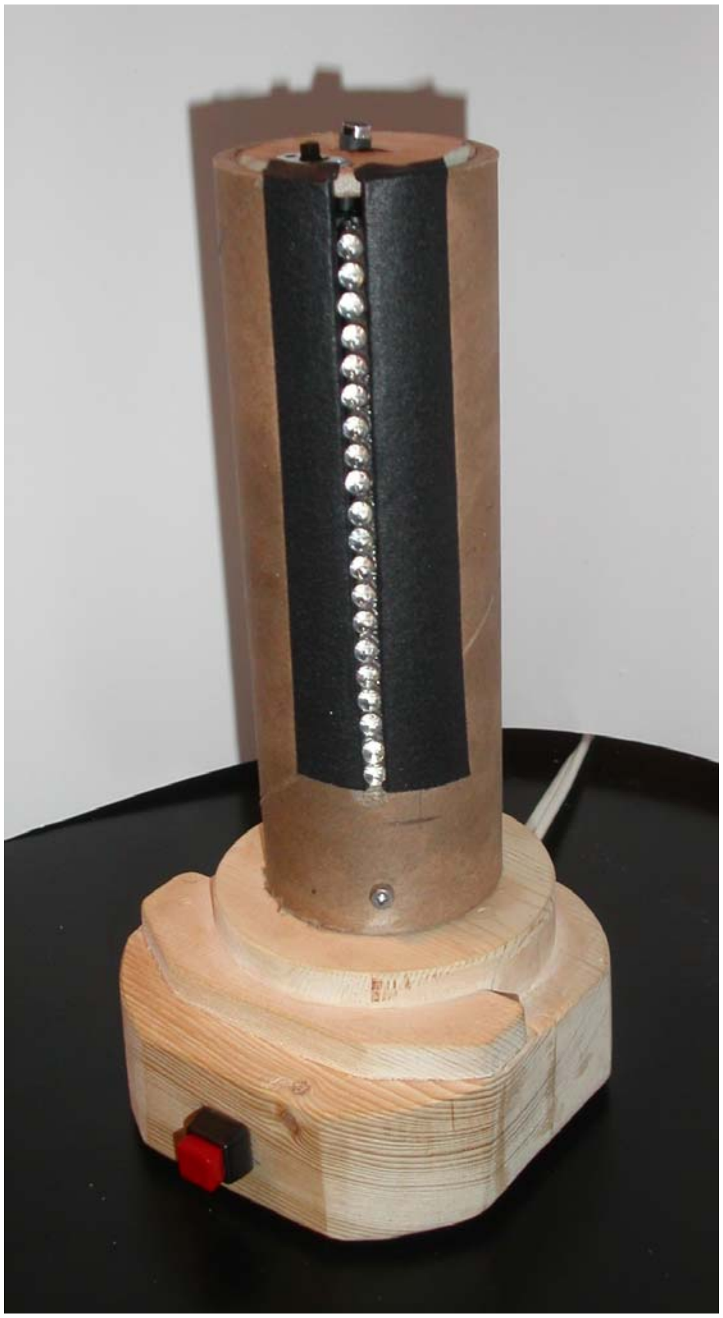
Appendix A – Hardware Schematic



Appendix B – PCB Layout Design



Appendix C – Pictures



Appendix D – Documented Source Code

```
/*===========================================================*\
                          MAIN

   Designed by: Andrew Boice & Jin Chen
   Date: 11/16/2005
\*===========================================================*/

#include "R_LED_V.h"
#include "ATD.h"
#include "SPI.h"
#include "LEDRoutine.h"

#pragma LINK_INFO DERIVATIVE "mc9s12c32"
#define DELAY       (20)
extern uchar RED_LEDS[NUM_LEDS];
extern uchar GREEN_LEDS[NUM_LEDS];
extern uchar BLUE_LEDS[NUM_LEDS];

void main(void) {
  uint i = 0;
  SetClockSpeed();

  initATD();
  initSPI();
  initLED();

  EnableInterrupts;

  for(;;)
  {
    initAlgo();
    for(i=0; i<DELAY; i++){}     //Delay
  }
}

void SetClockSpeed() {
// set clock speed to be 24MHz
  Crg.clksel.byte &=~PLLSEL; // disconnect PLL
  Crg.pllctl.byte |=PLLON;  // turn on PLL
  Crg.synr.byte |=SYN1; // set PLL multiplier
  Crg.refdv.byte |=REFDV0; //
  while(!(Crg.crgflg.byte & LOCK)){ };  // wait for PLL to lock in
  Crg.clksel.byte |=PLLSEL; // connect PLL
}
```

```
/*=================================================================== *\
                              Header for Main

  Designed by: Andrew Boice & Jin Chen
  Date: 11/16/2005
\*===================================================================*/

#ifndef R_LED_V_H       /*prevent duplicated includes*/
#define R_LED_V_H
#include <hidef.h>                          //Includes Code Warrior definitions
#include "per_C32_L45J.h"
#define Flash_Sector_Size 0x200

/*******************************************************************/
/*Macro Definitions                          (Defines below are functional examples)         */
/*******************************************************************/

#define int_enable()  {asm andcc   #0xEF;}    //interrupts enabled
#define int_disable() {asm orcc    #0x10;}    //interrupts disabled
#define wait()        {asm wait;}                //enter wait mode
#define stop_enable() {asm andcc   #0x7F;}    //stop mode enabled
#define stop()        {asm stop;}                //enter stop mode
#define nop()         {asm nop;}                 //enter NOP asm instruction
#define bgnd()        {asm bgnd; asm nop;}    //enter BGND asm instruction

#define ON 1                                                            //ON
#define OFF 0                                                           //OFF
#define ENABLE 1                                                        //Enable
#define DISABLE 1                                                       //DISABLE
#define TRUE 1                                                   //TRUE
#define FALSE 0                                                  //FALSE
#define PASS 0u                                                  //PASS
#define FAIL 1u                                                  //FAIL
#define SET 1u                                                   //SET
#define CLEAR 0u                                                        //CLEAR

#define CLI()   {asm cli;}                           //enable global interrupts
#define SEI()   {asm sei;}                            // disable global interrupts
#define NUL     0x00
#define ESC     0x1B
#define LEFT    0x4B
#define RIGHT   0x4D
#define HOME    0x47
#define END     0x4F
#define INSERT  0x52
#define DELETE  0x53
#define BACKSPC 0x08
#define ENTER   0x0D
#define CTLEND  0x75
#define CTLHOME 0x77
#define CTLRT   0x74
#define CTLLFT  0x73
void SetClockSpeed(void);
extern ulong busclk;
#define oscclk 24000                                         // Set Oscillator Freq. = 24 MHz
#endif //R_LED_V_H
```

```
/*=================================================================*\
                                    ATD

Description: Converts Analog output from the microphone to
                  8 bit digital values
Designed by: Andrew Boice & Jin Chen
Date: 11/16/2005
\*=================================================================*/

#include "ATD.h"

uchar getATD()
{
 return ATD_RESULT_0;
}

void initATD()
{
 ATD_CONTROL_2.bit.adpu = 1;         //Normal ATD functionality
 ATD_CONTROL_3.bit.slc = 0b0001;     //Sequence Length Coding = 1
 ATD_CONTROL_4.bit.sres8 = 1;        //Resolution = 8bit, select 0 for 10bit
 ATD_CONTROL_4.bit.prs = 0b00111;  //Prescaler = 16 Gives 8MHz
 ATD_CONTROL_5.bit.djm = 0;          //Result Register Data Justification = Left
                                            //Justified
 ATD_CONTROL_5.bit.dsgn = 0;                          //Unsigned Result Data
 ATD_CONTROL_5.bit.scan = 1;                          //Continuous Scan
 ATD_CONTROL_5.bit.mult = 0;                          //Sample only 1 channel.
 ATD_CONTROL_5.bit.cx = 0b000;                        //Sampling Channel 0
}
```

```
/*===================================================================*\
                              Header for ATD

 Designed by: Andrew Boice & Jin Chen
 Date: 11/16/2005
\*===================================================================*/

#ifndef ATD_H       /*prevent duplicated includes*/
#define ATD_H

#include "R_LED_V.h"

// ATD Register Definitions
#define ATD_CONTROL_2 Atd0.atdctl2
#define ATD_CONTROL_3 Atd0.atdctl3
#define ATD_CONTROL_4 Atd0.atdctl4
#define ATD_CONTROL_5 Atd0.atdctl5
#define ATD_STATUS_0  Atd0.atdstat0
#define ATD_STATUS_1  Atd0.atdstat1
#define ATD_PORT      Atd0.portad
#define ATD_RESULT_0  Atd0.atddr[0].d8.datah

// Function Prototypes
void initATD(void);
uchar getATD(void);

#endif //ATD_H
```

```
/*==============================================================*\
                              SPI

Description: Clocks out sequences of 8 bit data to LED Controllers
Designed by: Andrew Boice & Jin Chen
Date: 11/16/2005
\*==============================================================*/

#include "SPI.h"

void initSPI(void)
{
  SPI_CONTROL_1.bit.spe = 1;       //SPI Enabled
  SPI_CONTROL_1.bit.mstr = 1;      //SPI Master Mode
  SPI_CONTROL_1.bit.cpol = 1;      //Idle Clock State LOW
  SPI_CONTROL_2.bit.bidiroe = 1;  //Output Buffer Enabled
  SPI_BAUD_RATE.byte = 0x00;       //Baud Rate Divisor = 2
}

void sendSPI(uchar data)
{
  while(SPI_STATUS.bit.sptef == 0){}

  SPI_DATA_REG.byte = data;

  while(SPI_STATUS.bit.sptef == 0){}
}
```

```
/*==================================================================*\
                          Header for SPI

Designed by: Andrew Boice & Jin Chen
Date: 11/16/2005

\*==================================================================*/


#ifndef SPI_H        /*prevent duplicated includes*/
#define SPI_H

#include "R_LED_V.h"

// ATD Register Definitions
#define SPI_CONTROL_1 Spi0.spicr1
#define SPI_CONTROL_2 Spi0.spicr2
#define SPI_BAUD_RATE Spi0.spibr
#define SPI_STATUS    Spi0.spisr
#define SPI_DATA_REG  Spi0.spidr

// Function Prototypes
void initSPI(void);
void sendSPI(uchar data);

#endif //SPI_H
```

```
/*====================================================================*\
                              LEDRoutine

Description: Determine the location and control the on/off and color
                    status of the LEDs
Designed by: Andrew Boice & Jin Chen
Date: 11/16/2005
\*====================================================================*/

#include "LEDRoutine.h"

uchar LEDS[NUM_BYTE];

uchar RED_LEDS[NUM_LEDS];
uchar GREEN_LEDS[NUM_LEDS];
uchar BLUE_LEDS[NUM_LEDS];

void initLED()
{
  uchar i = 0;

  LATCH_PORT_DDR = 1;     //Set PTM3 as OUTPUT
  DRIVER_LATCH   = 0;     //Init Latch Enable to OFF

  for(i=0;i<NUM_BYTE;i++)           //Reset LEDs
  {
    LEDS[i] = 0;
  }

  for(i=0;i<NUM_LEDS;i++)           //Reset LEDs
  {
    RED_LEDS[i] = 0;
    BLUE_LEDS[i] = 0;
    GREEN_LEDS[i] = 0;
  }

}


void updateLED()                                          //update the status of LEDs
{
  uchar i;

  LEDS[0] = 0;

  if(BLUE_LEDS[15]){
    LEDS[0] |= 0b10000000;
  }
  if(GREEN_LEDS[16]){
    LEDS[0] |= 0b01000000;
  }
  if(RED_LEDS[16]){
    LEDS[0] |= 0b00100000;
  }
  if(GREEN_LEDS[17]){
    LEDS[0] |= 0b00010000;
  }
```

```
if(BLUE_LEDS[18]){
  LEDS[0] |= 0b00001000;
}
if(RED_LEDS[18]){
  LEDS[0] |= 0b00000100;
}
if(BLUE_LEDS[19]){
  LEDS[0] |= 0b00000010;
}


LEDS[1] = 0;

if(RED_LEDS[19]){
  LEDS[1] |= 0b10000000;
}
if(GREEN_LEDS[19]){
  LEDS[1] |= 0b01000000;
}
if(GREEN_LEDS[18]){
  LEDS[1] |= 0b00100000;
}
if(RED_LEDS[17]){
  LEDS[1] |= 0b00010000;
}
if(BLUE_LEDS[17]){
  LEDS[1] |= 0b00001000;
}
if(BLUE_LEDS[16]){
  LEDS[1] |= 0b00000100;
}
if(RED_LEDS[15]){
  LEDS[1] |= 0b00000010;
}
if(GREEN_LEDS[15]){
  LEDS[1] |= 0b00000001;
}


LEDS[2] = 0;

if(GREEN_LEDS[10]){
  LEDS[2] |= 0b10000000;
}
if(GREEN_LEDS[11]){
  LEDS[2] |= 0b01000000;
}
if(RED_LEDS[11]){
  LEDS[2] |= 0b00100000;
}
if(BLUE_LEDS[12]){
  LEDS[2] |= 0b00010000;
}
if(GREEN_LEDS[13]){
  LEDS[2] |= 0b00001000;
}
if(RED_LEDS[13]){
  LEDS[2] |= 0b00000100;
}
if(BLUE_LEDS[14]){
  LEDS[2] |= 0b00000010;
}


LEDS[3] = 0;

if(RED_LEDS[14]){
  LEDS[3] |= 0b10000000;
}
if(GREEN_LEDS[14]){
  LEDS[3] |= 0b01000000;
}
if(BLUE_LEDS[13]){
  LEDS[3] |= 0b00100000;
}
if(RED_LEDS[12]){
  LEDS[3] |= 0b00010000;
}
if(GREEN_LEDS[12]){
  LEDS[3] |= 0b00001000;
}
if(BLUE_LEDS[11]){
  LEDS[3] |= 0b00000100;
}
if(RED_LEDS[10]){
  LEDS[3] |= 0b00000010;
}
if(BLUE_LEDS[10]){
  LEDS[3] |= 0b00000001;
}


LEDS[4] = 0;

if(BLUE_LEDS[5]){
  LEDS[4] |= 0b10000000;
}
if(GREEN_LEDS[6]){
  LEDS[4] |= 0b01000000;
}
if(RED_LEDS[6]){
  LEDS[4] |= 0b00100000;
}
if(BLUE_LEDS[7]){
  LEDS[4] |= 0b00010000;
}
if(GREEN_LEDS[8]){
  LEDS[4] |= 0b00001000;
}
if(RED_LEDS[8]){
  LEDS[4] |= 0b00000100;
}
if(GREEN_LEDS[9]){
  LEDS[4] |= 0b00000010;
```

```
}


LEDS[5] = 0;

if(RED_LEDS[9]){
  LEDS[5] |= 0b10000000;
}
if(BLUE_LEDS[9]){
  LEDS[5] |= 0b01000000;
}
if(BLUE_LEDS[8]){
  LEDS[5] |= 0b00100000;
}
if(RED_LEDS[7]){
  LEDS[5] |= 0b00010000;
}
if(GREEN_LEDS[7]){
  LEDS[5] |= 0b00001000;
}
if(BLUE_LEDS[6]){
  LEDS[5] |= 0b00000100;
}
if(RED_LEDS[5]){
  LEDS[5] |= 0b00000010;
}
if(GREEN_LEDS[5]){
  LEDS[5] |= 0b00000001;
}


LEDS[6] = 0;

if(GREEN_LEDS[0]){
  LEDS[6] |= 0b10000000;
}
if(BLUE_LEDS[1]){
  LEDS[6] |= 0b01000000;
}
if(RED_LEDS[1]){
  LEDS[6] |= 0b00100000;
}
if(BLUE_LEDS[2]){
  LEDS[6] |= 0b00010000;
}
if(BLUE_LEDS[3]){
  LEDS[6] |= 0b00001000;
}
if(RED_LEDS[3]){
  LEDS[6] |= 0b00000100;
}
if(BLUE_LEDS[4]){
  LEDS[6] |= 0b00000010;
}


LEDS[7] = 0;
```

```
  if(RED_LEDS[4]){
    LEDS[7] |= 0b10000000;
  }
  if(GREEN_LEDS[4]){
    LEDS[7] |= 0b01000000;
  }
  if(GREEN_LEDS[3]){
    LEDS[7] |= 0b00100000;
  }
  if(RED_LEDS[2]){
    LEDS[7] |= 0b00010000;
  }
  if(GREEN_LEDS[2]){
    LEDS[7] |= 0b00001000;
  }
  if(GREEN_LEDS[1]){
    LEDS[7] |= 0b00000100;
  }
  if(RED_LEDS[0]){
    LEDS[7] |= 0b00000010;
  }
  if(BLUE_LEDS[0]){
    LEDS[7] |= 0b00000001;
  }

  //sendSPI(0x00);

  for(i=0; i<NUM_BYTE; i++)
  {
    sendSPI(LEDS[i]);
  }

  latchData();

}
void latchData()          //turn on the LEDs
{
  uint i;

  for(i=0; i<LATCH_DELAY; i++){}    //Delay between latching intervals

  DRIVER_LATCH = 1;

  for(i=0; i<LATCH_DELAY; i++){}    //Delay between latching intervals

  DRIVER_LATCH = 0;
}
```

```
/*==================================================================*\
                               Header for LED_ROUTINE

 Designed by: Andrew Boice & Jin Chen
 Date: 11/16/2005
\*==================================================================*/

#ifndef LED_ROUTINE_H        /*prevent duplicated includes*/
#define LED_ROUTINE_H

#include "SPI.h"

// LEDRoutine Register Definitions
#define NUM_BYTE        (8)
#define DRIVER_LATCH    (Pim.ptm.bit.ptm3)
#define LATCH_PORT_DDR  (Pim.ddrm.bit.ddrm3)
#define LATCH_DELAY     (10)
#define NUM_LEDS        (20)


// Function Prototypes
void initLED(void);
void updateLED(void);
void latchData(void);

#endif //LED_ROUTINE_H
```

```
/*=====================================================================*\
                                  Algorithems

Description: Governs the theme of the LED
Designed by: Andrew Boice & Jin Chen
Date: 11/16/2005
\*=====================================================================*/
#include"Algorithems.h"

#define NOISE_FLOOR (15)
#define INCREMENT   (2)
#define NUM_TO_AVG  (1000)
#define DECAY       (1)
#define RED_DECAY   (10)
#define HANG_TIME   (40)

extern uchar RED_LEDS[NUM_LEDS];
extern uchar GREEN_LEDS[NUM_LEDS];
extern uchar BLUE_LEDS[NUM_LEDS];

uint Volume = 0;
uint Counter = 0;
ulong AccumulatedVolume = 0;
uint OutputVolume = 0;
uint RedLevel = 0;
uint CurrentRedLevel = 0;
uint HangCounter = 0;

void collectVolume(void);
void averageVolume(void);
void decayVolume(void);
void executeAlgo(void);
void algoEnhanced(void);

void initAlgo()                    //initialize algorthim by calling set of instructions
{
                              //  defined in the code
  if(Counter < NUM_TO_AVG)
  {
    collectVolume();
    Counter++;
  }
  else
  {
    averageVolume();
    decayVolume();
    executeAlgo();
    Counter = 0;
    AccumulatedVolume = 0;
  }

}



void collectVolume()
{
  Volume = getATD();

  if(Volume > 123)
  {
   Volume = Volume - 123;
  }
  else
  {
   Volume = 123 - Volume;   //Digital rectification of the ATD result
  }

  AccumulatedVolume = Volume + AccumulatedVolume;
}


void averageVolume()                                        //Average several readings into 1
{

  Volume = AccumulatedVolume/NUM_TO_AVG;
}


void executeAlgo()                                                  //Execute the specified LED
theme algorithm
{
  //algoStandard();
  algoEnhanced();
}



void decayVolume()          //Implement the decay effect of the LED
{
  if(OutputVolume <= Volume)
  {
    OutputVolume = Volume;
  }
  else
  {
    if((OutputVolume - Volume) > DECAY)
    {
     OutputVolume = OutputVolume - DECAY;
    }
    else
    {
     OutputVolume = Volume;
    }

  }

}


void algoEnhanced()
{
   uint i = 0;
```

```
for(i=0;i<NUM_LEDS;i++)  //Clear LEDs
 {
   RED_LEDS[i] = 0;
   BLUE_LEDS[i] = 1;
   GREEN_LEDS[i] = 0;
 }

 if(OutputVolume > NOISE_FLOOR)
 {
   GREEN_LEDS[0] = 1;
   BLUE_LEDS[0] = 0;
 }
 RedLevel =0 ;

 if(OutputVolume > NOISE_FLOOR + INCREMENT)
 {
   GREEN_LEDS[1] = 1;
   BLUE_LEDS[1] = 0;
   RedLevel = 100;
 }

 if(OutputVolume > NOISE_FLOOR + 2*INCREMENT)
 {
   GREEN_LEDS[2] = 1;
   BLUE_LEDS[2] = 0;
   //RED_LEDS[1] = 0;
   //RED_LEDS[2] = 1;
   RedLevel = 200;
 }

 if(OutputVolume > NOISE_FLOOR + 3*INCREMENT)
 {
   GREEN_LEDS[3] = 1;
   BLUE_LEDS[3] = 0;
   RedLevel = 300;
 }

 if(OutputVolume > NOISE_FLOOR + 4*INCREMENT)
 {
   GREEN_LEDS[4] = 1;
   BLUE_LEDS[4] = 0;
   RedLevel = 400;
 }

 if(OutputVolume > NOISE_FLOOR + 5*INCREMENT)
 {
   GREEN_LEDS[5] = 1;
   BLUE_LEDS[5] = 0;
   RedLevel = 500;
 }

 if(OutputVolume > NOISE_FLOOR + 6*INCREMENT)
 {
   GREEN_LEDS[6] = 1;
   BLUE_LEDS[6] = 0;
   RedLevel = 600;
 }
```

```
if(OutputVolume > NOISE_FLOOR + 7*INCREMENT)
{
 GREEN_LEDS[7] = 1;
 BLUE_LEDS[7] = 0;
 RedLevel = 700;
}

if(OutputVolume > NOISE_FLOOR + 8*INCREMENT)
{
 GREEN_LEDS[8] = 1;
 BLUE_LEDS[8] = 0;
 //RED_LEDS[7] = 0;
 //RED_LEDS[8] = 1;
 RedLevel = 800;
}

if(OutputVolume > NOISE_FLOOR + 9*INCREMENT)
{
 GREEN_LEDS[9] = 1;
 BLUE_LEDS[9] = 0;
 RedLevel = 900;
}

if(OutputVolume > NOISE_FLOOR + 10*INCREMENT)
{
 GREEN_LEDS[10] = 1;
 BLUE_LEDS[10] = 0;
 RedLevel = 1000;
}

if(OutputVolume > NOISE_FLOOR + 11*INCREMENT)
{
 GREEN_LEDS[11] = 1;
 BLUE_LEDS[11] = 0;
 RedLevel = 1100;
}

if(OutputVolume > NOISE_FLOOR + 12*INCREMENT)
{
 GREEN_LEDS[12] = 1;
 BLUE_LEDS[12] = 0;
 RedLevel = 1200;
}

if(OutputVolume > NOISE_FLOOR + 13*INCREMENT)
{
 GREEN_LEDS[13] = 1;
 BLUE_LEDS[13] = 0;
 RedLevel = 1300;
}

if(OutputVolume > NOISE_FLOOR + 14*INCREMENT)
{
 GREEN_LEDS[14] = 1;
 BLUE_LEDS[14] = 0;
 RedLevel = 1400;
}

if(OutputVolume > NOISE_FLOOR + 15*INCREMENT)
{
 GREEN_LEDS[15] = 1;
 BLUE_LEDS[15] = 0;
 RedLevel = 1500;
}

if(OutputVolume > NOISE_FLOOR + 16*INCREMENT)
{
 GREEN_LEDS[16] = 1;
 BLUE_LEDS[16] = 0;
 RedLevel = 1600;
}

if(OutputVolume > NOISE_FLOOR + 17*INCREMENT)
{
 GREEN_LEDS[17] = 1;
 BLUE_LEDS[17] = 0;
 RedLevel = 1700;
}

if(OutputVolume > NOISE_FLOOR + 18*INCREMENT)
{
 GREEN_LEDS[18] = 1;
 BLUE_LEDS[18] = 0;
 RedLevel = 1800;
}

if(OutputVolume > NOISE_FLOOR + 19*INCREMENT)
{
 GREEN_LEDS[19] = 1;
 BLUE_LEDS[19] = 0;
 RedLevel = 1900;
}

if(CurrentRedLevel <= RedLevel)
{
 CurrentRedLevel = RedLevel;
 HangCounter = 0;
}
else
{
 if(HangCounter < HANG_TIME)
 {
   HangCounter++;
 }
 else
 {
  CurrentRedLevel = CurrentRedLevel - RED_DECAY;
 }


}

RedLevel = CurrentRedLevel/100;
```

```
  if(CurrentRedLevel%100 > 50)
  {
    RedLevel++;
  }

  if(RedLevel != 0)
  {
    GREEN_LEDS[RedLevel] = 0;
    BLUE_LEDS[RedLevel] = 0;
    RED_LEDS[RedLevel] = 1;
  }
  updateLED();
}
```

```
/*=================================================================*\
                              Headers for Algorithems

 Designed by: Andrew Boice & Jin Chen
 Date: 11/16/2005
\*=================================================================*/

#ifndef Algorithems_H        /*prevent duplicated includes*/
#define Algorithems_H

#include "R_LED_V.h"
#include "LEDRoutine.h"
#include "ATD.h"

// Algorithems Register Definitions


// Function Prototypes
void initAlgo(void);
void algoStandard(void);

#endif //Algorithems_H
```