Lab 11 Final Embedded System

Duc Tran & Brandon Wong

12/6/2013

1.0 Overview

1.1 Objectives: Why are we doing this project? What is the purpose?

The objective is to combine our knowledge and tools from our previous labs and to create a final embedded system project. We chose to do a 5x5x5 LED cube equalizer. Educationally students are learning the issue of power management, clock , reset, programming embedded system, cost efficiency, and layout of PCB

1.2 Roles and Responsibilities: Who will do what Who are the clients?

The TA are our clients, and Duc and Brandon are the engineers. Duc will be modifying the code he wrote previously for the music FSM and switch interface, and Systick interface. Additionally, Duc will be adding code used for the GPIO for the outputs of the LM3S811. Duc and Brandon will both come up with visual effects and ideas for the LED cube. Brandon will help revise code as well as assembling the 5x5x5 and soldering components to the PCB. Together, we will modify the design of LED PCB and the LM3S811 schematic.

2.0 Function Description

2.1 Functionality: What will the system do precisely?

A music player will be set up with the LED cube via audio jack. When music is played, the LEDs on the cube will make a visual effect as the music is playing. Each LED will turn on depending on the frequencies the music is playing. The audio will be connected via audio splitter so that the LED cube can take in analog signals, and also users can hear the music and see the visual effects. Instead of the LED cube being powered by a battery, we will use a wall outlet to power the LED cube due to the amount of current for 125 LEDs was impossible for finding a cheap and long lasting battery.

2.2 Performance: Define the measures and describe how they will be determined.

The system will be judged by three qualitative measures. First, the software modules must be easy to understand and well organized. Second, the system must employ a finite state machine running in the background. There should be a clear and obvious abstraction, separating what the machine does (the FSM state diagram) from how the machine works (the software ISR). Third, all software will be judged according to style guidelines. There are three quantitative measures. First, we must measure the frequency at what toggles the LED. Second, the maximum time to run one instance of the ISR will be recorded. Third, you will measure power supply current to run the system.

2.3 Usability: Describe the interfaces. Be quantitative if possible.

There will be 125 LEDs, five 3-8 decoders that will multiplex each individual LED, and an audio jack that will read the music. An LED will be mapped onto an output of an decoder, and the 5th decoder will be the selector of which layer you want to activate. When the ADC samples a frequency, the ADC converts it into a voltage. The voltage can tell us what frequency and which LEDs we want to activate onto the cube. The effects created are up to the us, and we can generate infinite amounts of effects. The idea is that we can create which LEDs we want to activate, create an array for the selected LEDs, and use a random function to iterate random LEDs to achieve an effect.

3.0 Deliverables

3.1 Reports

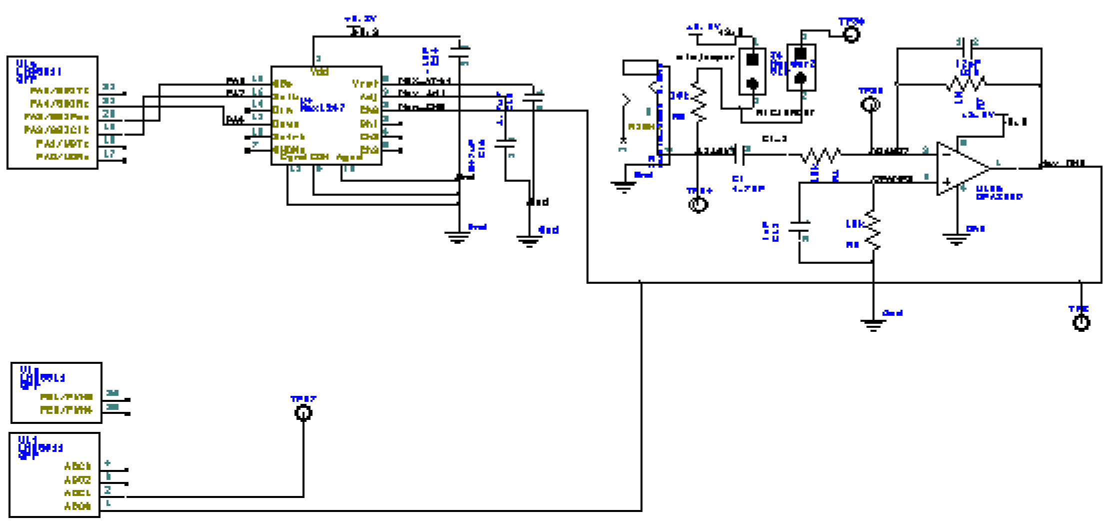
A lab report described below is due by December 6, 2013. This report includes the final requirements document.

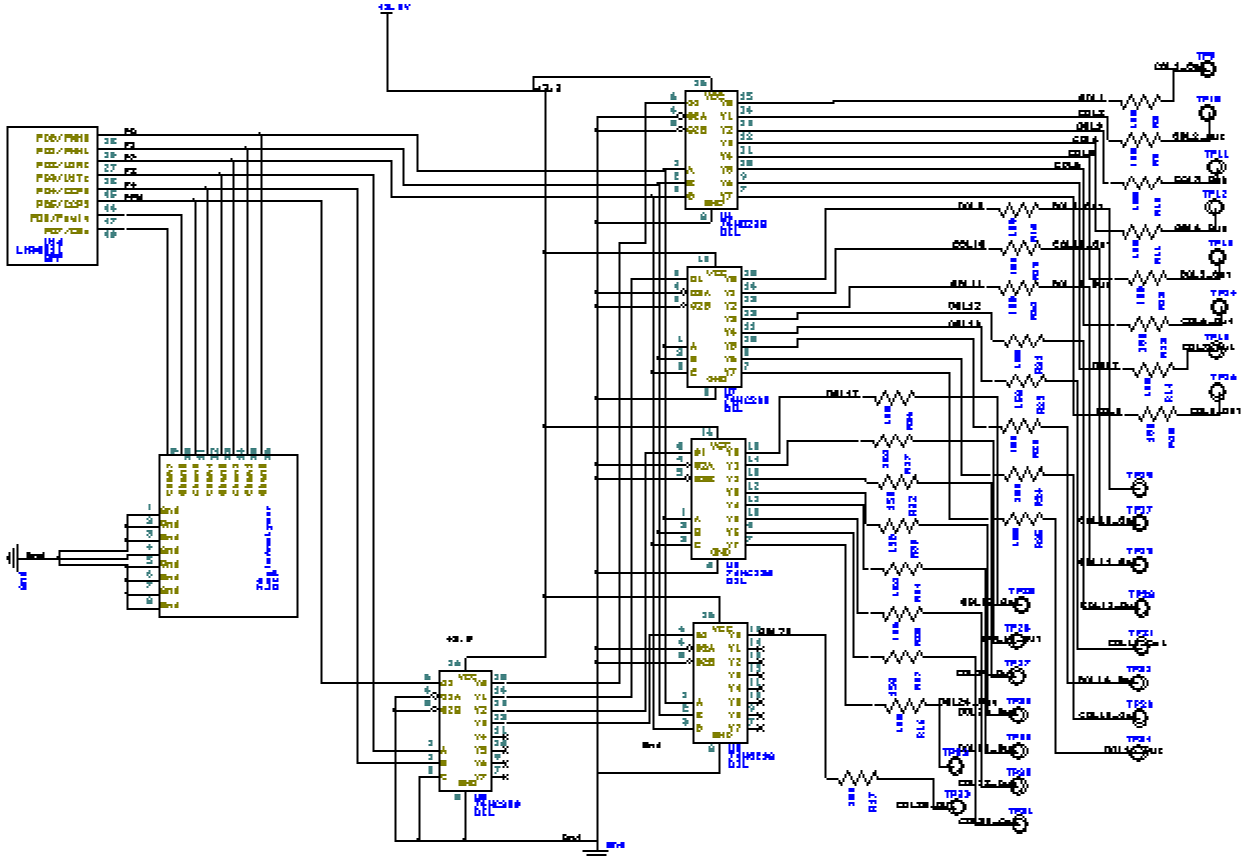
3.2 Outcomes

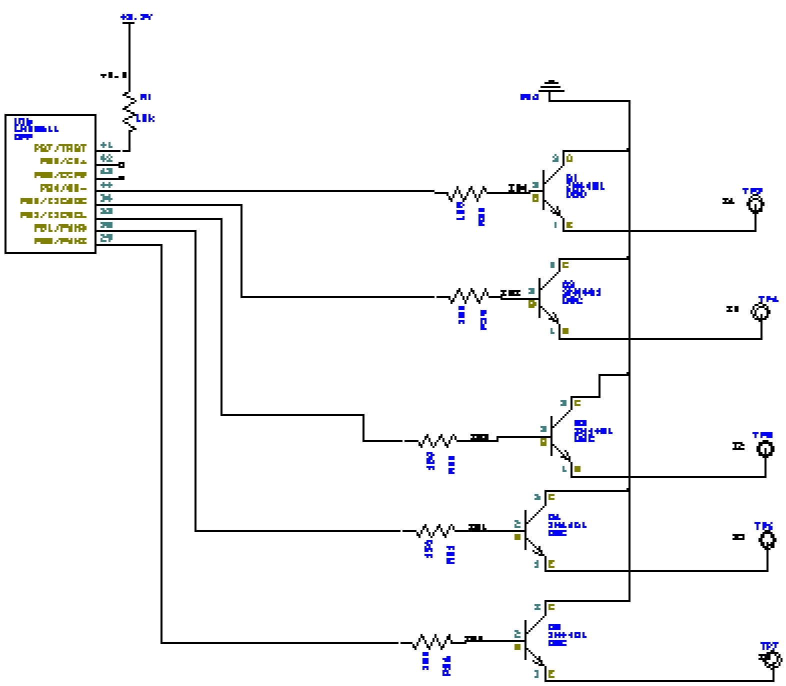
There are three deliverables: demonstration, exhibit, and report. If selected during demonstration, then we get to present our project in front of a group of judges on December 6, 2013.

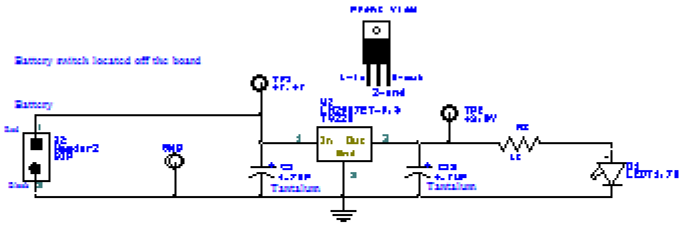
4.0 Hardware Design

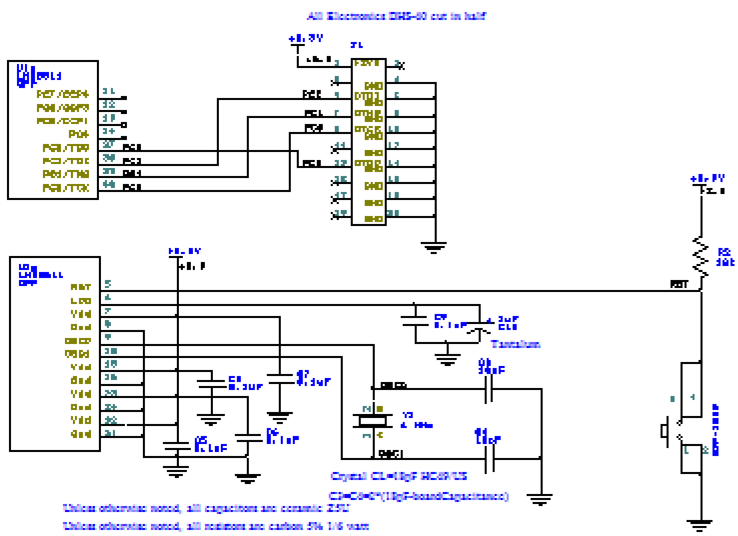
Modified circuit diagram (SCH file)



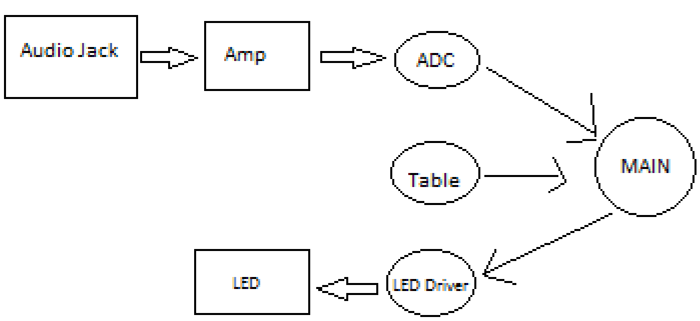








5.0 Software Design



The audio jack collects analog signals from the music. The amplifier will collect it and convert it into a digital signal. Whenever we get a voltage, we can look up a table of arrays of effects and output them to the LEDs.

6.0 Measurement Data
We included at least 25 test points for each of the LED columns to see if voltage is in the right places at the right time. The test points in various parts of the PCB can tell us if it is getting a supply voltage of 3.3 volts. We can also use a multimeter to measure the voltage across each LED if we expect a voltage on a given LED.

7.0 Analysis and Discussion
The construction and design of our 5x5x5 LED cube equalizer turned out successful. Initially some of the LEDs on the cube were not working, but we realized through measurements that some of the LEDs were unconnected and therefore leaving a column of LEDs unlit. The software and the effects were designed by us, and of course there is an infinite amount of effects that can be achieved. However, everything worked as in the software can read the ADC value and you can see the effects change within the beat of the music. We did not end up using a battery for the LED because the amount of current from the LEDs require a long lasting battery that was expensive even though battery is not within our budget. Instead, we powered it by the LM3S1968, and the LM3S1968 was powered by a USB connected to a wall outlet. In the future, we hope to create more effects for the LED driver so that users are more entertained by the visualizations.

Code

```
//Filename: GPIO.c
//Author: Duc Tran, Brandon Wong
//Initial Creation Date: November 4, 2013
//Description: General Port Input/Output that controls LED
//Lab Number: W 2-3:30
//TA : Omar & Mahesh
//Date of last revision : December 6, 2013
//Hardware Configuration : NONE

// GPIO.c
// Runs on LM3S811
// Initialize four GPIO pins as outputs.  Continually generate output to
// drive simulated stepper motor.
// Daniel Valvano
// July 11, 2011

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
   Example 2.2, Program 2.8, Figure 2.27
```

```
#include "LEDS_Driver.h"
#include "Pattern_Generator.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "Systick.h"
#include <stdlib.h>
#include <stdio.h>
// PD3 is an output to LED3, negative logic
// PD2 is an output to LED2, negative logic
// PD1 is an output to LED1, negative logic
// PD0 is an outpSut to LED0, negative logic

#define GPIO_PORTD_DATA_R       (*((volatile unsigned long *)0x400073FC))
#define GPIO_PORTD_DIR_R        (*((volatile unsigned long *)0x40007400))
#define GPIO_PORTD_AFSEL_R      (*((volatile unsigned long *)0x40007420))
#define GPIO_PORTD_DEN_R        (*((volatile unsigned long *)0x4000751C))
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC2_GPIOD      0x00000008  // port D Clock Gating Control
#define LEDS                    (*((volatile unsigned long *)0x4000703C))

//-----------------------------------------------------------------------------------------------------------
--------//
```

```
// Sine wave from signal generator connected to ADC1
//
// This program periodically samples ADC channel 1 and stores the
// result to a buffer.  After the buffer is full, it stops
// triggering ADC conversions and outputs the results to the UART
// separated by TABs.



#define NVIC_EN0_INT17          0x00020000  // Interrupt 17 enable
#define NVIC_EN0_R              (*((volatile unsigned long *)0xE000E100))  // IRQ 0 to 31 Set Enable
Register
#define NVIC_PRI4_R             (*((volatile unsigned long *)0xE000E410))  // IRQ 16 to 19 Priority
Register
#define TIMER0_CFG_R            (*((volatile unsigned long *)0x40030000))
#define TIMER0_TAMR_R           (*((volatile unsigned long *)0x40030004))
#define TIMER0_CTL_R            (*((volatile unsigned long *)0x4003000C))
#define TIMER0_IMR_R            (*((volatile unsigned long *)0x40030018))
#define TIMER0_TAILR_R          (*((volatile unsigned long *)0x40030028))
#define TIMER0_TAPR_R           (*((volatile unsigned long *)0x40030038))
#define TIMER_CFG_16_BIT        0x00000004  // 16-bit timer configuration,
                                // function is controlled by bits
                                // 1:0 of GPTMTAMR and GPTMTBMR
#define TIMER_TAMR_TAMR_PERIOD  0x00000002  // Periodic Timer mode
#define TIMER_CTL_TAOTE         0x00000020  // GPTM TimerA Output Trigger
                                // Enable
#define TIMER_CTL_TAEN          0x00000001  // GPTM TimerA Enable
#define TIMER_IMR_TATOIM        0x00000001  // GPTM TimerA Time-Out Interrupt
                                // Mask
#define TIMER_TAILR_TAILRL_M    0x0000FFFF  // GPTM TimerA Interval Load
                                // Register Low
#define ADC_ACTSS_R             (*((volatile unsigned long *)0x40038000))
#define ADC0_RIS_R              (*((volatile unsigned long *)0x40038004))
#define ADC0_IM_R               (*((volatile unsigned long *)0x40038008))
#define ADC0_ISC_R              (*((volatile unsigned long *)0x4003800C))
#define ADC0_EMUX_R             (*((volatile unsigned long *)0x40038014))
#define ADC0_SSPRI_R            (*((volatile unsigned long *)0x40038020))
#define ADC0_PSSI_R             (*((volatile unsigned long *)0x40038028))
#define ADC0_SSMUX3_R           (*((volatile unsigned long *)0x400380A0))
```

```c
#define ADC0_SSCTL3_R           (*((volatile unsigned long *)0x400380A4))
#define ADC0_SSFIFO3_R          (*((volatile unsigned long *)0x400380A8))
#define ADC_ACTSS_ASEN3         0x00000008  // ADC SS3 Enable
#define ADC_RIS_INR3            0x00000008  // SS3 Raw Interrupt Status
#define ADC_IM_MASK3            0x00000008  // SS3 Interrupt Mask
#define ADC_ISC_IN3             0x00000008  // SS3 Interrupt Status and Clear
#define ADC_EMUX_EM3_M          0x0000F000  // SS3 Trigger Select mask
#define ADC_EMUX_EM3_TIMER      0x00005000  // Timer
#define ADC_SSPRI_SS3_4TH       0x00003000  // fourth priority
#define ADC_SSPRI_SS2_3RD       0x00000200  // third priority
#define ADC_SSPRI_SS1_2ND       0x00000010  // second priority
#define ADC_SSPRI_SS0_1ST       0x00000000  // first priority
#define ADC_PSSI_SS3            0x00000008  // SS3 Initiate
#define ADC_SSMUX3_MUX0_M       0x00000003  // 1st Sample Input Select mask
#define ADC_SSMUX3_MUX0_S       0           // 1st Sample Input Select lshift
#define ADC_SSCTL3_TS0          0x00000008  // 1st Sample Temp Sensor Select
#define ADC_SSCTL3_IE0          0x00000004  // 1st Sample Interrupt Enable
#define ADC_SSCTL3_END0         0x00000002  // 1st Sample is End of Sequence
#define ADC_SSCTL3_D0           0x00000001  // 1st Sample Diff Input Select
#define ADC_SSFIFO3_DATA_M      0x000003FF  // Conversion Result Data mask
#define GPIO_PORTA_AFSEL_R      (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTC_DATA_R       (*((volatile unsigned long *)0x400063FC))
#define GPIO_PORTC_DIR_R        (*((volatile unsigned long *)0x40006400))
#define GPIO_PORTC_DEN_R        (*((volatile unsigned long *)0x4000651C))
#define SYSCTL_RCGC0_R          (*((volatile unsigned long *)0x400FE100))
#define SYSCTL_RCGC1_R          (*((volatile unsigned long *)0x400FE104))
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
#define SYSCTL_RCGC0_ADC        0x00010000  // ADC0 Clock Gating Control
#define SYSCTL_RCGC0_ADCSPD_M   0x00000300  // ADC Sample Speed mask
#define SYSCTL_RCGC0_ADCSPD500K 0x00000200  // 500K samples/second
#define SYSCTL_RCGC1_TIMER0     0x00010000  // timer 0 Clock Gating Control
#define SYSCTL_RCGC2_GPIOC      0x00000004  // port C Clock Gating Control
#define SYSCTL_RCGC2_GPIOA      0x00000001  // port A Clock Gating Control
#define MAXBUFFERSIZE           50          // maximum number of samples
#define SAMPLEFREQ              1000        // sampling frequency (min. 92 Hz)
#define CLOCKFREQ               6000000     // default clock frequency


void DisableInterrupts(void); // Disable interrupts
```

```
void EnableInterrupts(void);  // Enable interrupts
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value
void WaitForInterrupt(void);  // low power mode

// There are many choices to make when using the ADC, and many
// different combinations of settings will all do basically the
// same thing.  For simplicity, this function makes some choices
// for you.  When calling this function, be sure that it does
// not conflict with any other software that may be running on
// the microcontroller.  Particularly, ADC sample sequencer 3
// is used here because it only takes one sample, and only one
// sample is absolutely needed.  Sample sequencer 3 generates a
// raw interrupt when the conversion is complete, and it is then
// promoted to an ADC controller interrupt.  Hardware Timer0A
// triggers the ADC conversion at the programmed interval, and
// software handles the interrupt to process the measurement
// when it is complete.
//
// A simpler approach would be to use software to trigger the
// ADC conversion, wait for it to complete, and then process the
// measurement.
//
// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Timer0A: enabled
// Mode: 16-bit, down counting
// One-shot or periodic: periodic
// Prescale value: programmable using variable 'prescale' [0:255]
// Interval value: programmable using variable 'period' [0:65535]
// Sample time is busPeriod*(prescale+1)*(period+1)
// Max sample rate: <=500,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: Timer0A
// SS3 1st sample source: programmable using variable 'channelNum' [0:3]
```

```
// SS3 interrupts: enabled and promoted to controller

/****************ADC_InitTimer0ATriggerSeq3***************
 Description: Initializes ADC Interrupt
 range 0 to 999.99
 Input: unsigned char channelNum, unsigned char prescale, unsigned short period
 Output: none
   */
void ADC_InitTimer0ATriggerSeq3(unsigned char channelNum, unsigned char prescale, unsigned short
period){
  volatile unsigned long delay;
  // channelNum must be 0-3 (inclusive) corresponding to ADC0 through ADC3
  if(channelNum > 3){
    return;                              // invalid input, do nothing
  }
  DisableInterrupts();
  // **** general initialization ****
  SYSCTL_RCGC0_R |= SYSCTL_RCGC0_ADC;       // activate ADC
  SYSCTL_RCGC0_R &= ~SYSCTL_RCGC0_ADCSPD_M; // clear ADC sample speed field
  SYSCTL_RCGC0_R += SYSCTL_RCGC0_ADCSPD500K;// configure for 500K ADC max
sample rate
  SYSCTL_RCGC1_R |= SYSCTL_RCGC1_TIMER0;    // activate timer0
  delay = SYSCTL_RCGC1_R;                   // allow time to finish activating
  TIMER0_CTL_R &= ~TIMER_CTL_TAEN;          // disable timer0A during setup
  TIMER0_CTL_R |= TIMER_CTL_TAOTE;          // enable timer0A trigger to ADC
  TIMER0_CFG_R = TIMER_CFG_16_BIT;          // configure for 16-bit timer mode
  // **** timer0A initialization ****
  TIMER0_TAMR_R = TIMER_TAMR_TAMR_PERIOD;   // configure for periodic mode
  TIMER0_TAPR_R = prescale;                 // prescale value for trigger
  TIMER0_TAILR_R = period;                  // start value for trigger
  TIMER0_IMR_R &= ~TIMER_IMR_TATOIM;        // disable timeout (rollover) interrupt
  TIMER0_CTL_R |= TIMER_CTL_TAEN;           // enable timer0A 16-b, periodic, no interrupts
  // **** ADC initialization ****
                                 // sequencer 0 is highest priority (default setting)
                                 // sequencer 1 is second-highest priority (default setting)
                                 // sequencer 2 is third-highest priority (default setting)
                                 // sequencer 3 is lowest priority (default setting)
  ADC0_SSPRI_R =
(ADC_SSPRI_SS0_1ST|ADC_SSPRI_SS1_2ND|ADC_SSPRI_SS2_3RD|ADC_SSPRI_SS3_4T
```

```
H);
  ADC_ACTSS_R &= ~ADC_ACTSS_ASEN3;          // disable sample sequencer 3
  ADC0_EMUX_R &= ~ADC_EMUX_EM3_M;           // clear SS3 trigger select field
  ADC0_EMUX_R += ADC_EMUX_EM3_TIMER;        // configure for timer trigger event
  ADC0_SSMUX3_R &= ~ADC_SSMUX3_MUX0_M;      // clear SS3 1st sample input select field
                                // configure for 'channelNum' as first sample input
  ADC0_SSMUX3_R += (channelNum<<ADC_SSMUX3_MUX0_S);
  ADC0_SSCTL3_R = (0                        // settings for 1st sample:
              & ~ADC_SSCTL3_TS0        // read pin specified by ADC0_SSMUX3_R (default
setting)
              | ADC_SSCTL3_IE0         // raw interrupt asserted here
              | ADC_SSCTL3_END0        // sample is end of sequence (default setting, hardwired)
              & ~ADC_SSCTL3_D0);       // differential mode not used (default setting)
  ADC0_IM_R |= ADC_IM_MASK3;                // enable SS3 interrupts
  ADC_ACTSS_R |= ADC_ACTSS_ASEN3;           // enable sample sequencer 3
  // **** interrupt initialization ****
                                // ADC3=priority 2
  NVIC_PRI4_R = (NVIC_PRI4_R&0xFFFF00FF)|0x00004000; // bits 13-15
  NVIC_EN0_R |= NVIC_EN0_INT17;             // enable interrupt 17 in NVIC
  EnableInterrupts();
}
volatile unsigned short index = 0;
volatile unsigned long ADCbuffer[MAXBUFFERSIZE];

/***************ADC3_Handler***************
 Description: ADC Handler
 Input: none
 Output: none
  */
void ADC3_Handler(void){
  ADC0_ISC_R = ADC_ISC_IN3;             // acknowledge ADC sequence 3 completion
  ADCbuffer[index] = ADC0_SSFIFO3_R&ADC_SSFIFO3_DATA_M;
  index = index + 1;
  if(index == MAXBUFFERSIZE){
    ADC_ACTSS_R &= ~ADC_ACTSS_ASEN3;    // disable sample sequencer 3
    TIMER0_CTL_R &= ~TIMER_CTL_TAEN;    // disable timer0A
  }
}
```

```
//---------------------------------------------------------------------------------------------------
--------//
// access PD3-PD0
// delay function for testing from sysctl.c
// which delays 3*ulCount cycles
#ifdef __TI_COMPILER_VERSION__
	//Code Composer Studio Code
	void Delay(unsigned long ulCount){
	__asm (	"    subs    r0, #1\n"
			"    bne     Delay\n"
			"    bx      lr\n");
}

#else
	//Keil uVision Code
	__asm void
	Delay(unsigned long ulCount)
	{
   subs    r0, #1
   bne     Delay
   bx      lr
	}

#endif


unsigned short const ADCdata[53]={0,8,6,21,37,53,69,85,102,119,136,
     153,171,188,206,225,243,262,281,300,319,
     339,359,379,400,421,442,463,484,506,528,
     550,573,596,619,642,666,689,713,738,762,
     787,812,837,863,889,915,941,967,994,1021,1023,1024};
unsigned short const Tdata[53]={4000,4000,3960,3920,3880,3840,3800,3760,3720,3680,3640,

     3600,3560,3520,3480,3440,3400,3360,3320,3280,3240,
```

```
    3200,3160,3120,3080,3040,3000,2960,2920,2880,2840,
    2800,2760,2720,2680,2640,2600,2560,2520,2480,2440,
    2400,2360,2320,2280,2240,2200,2160,2120,2080,2040,2000,2000};

/***************Find_ADCindex**************
 Description: Finds Index of ADC value
 Input: unsigned short ADC
 Output: index of ADC
   */
int Find_ADCindex(unsigned short ADC)
{
   int index1 =0;
   int min = 1024;
   int rindex = 0;
   for(index1 = 0; index1 < 53; index1++)
   {
      if((ADCdata[index1]<ADC))
      {
         rindex = index1;
      }
   }

   return rindex;
}


int main(void){  volatile unsigned long delay;
         int i,j,state,intensity,counter;
         SysTick_Init();
         LED_Init();
         counter = 0;
         state = 1;
         Alphabet('T');
         Alphabet('E');
   Alphabet('X');
         Alphabet('A');
         Alphabet('S');
         for (i =0;i < 25;i++){
                  SpinUT();
```

```
}
while(1){
	index = 0;
	ADC_InitTimer0ATriggerSeq3(0, 0, CLOCKFREQ/SAMPLEFREQ);
	while(index < MAXBUFFERSIZE){};
	for(i=1; i<MAXBUFFERSIZE; i=i+1){
		Turn_Off();
		if(ADCbuffer[i] > 8){
			intensity = Find_ADCindex(ADCbuffer[i]);
			switch (state){
				case 0://welcome
					Animation0(Tdata[intensity]-1000);
					if(counter > 200){
						state = 1;
						counter = 0;
					}else{counter = counter +1;}
				break;
				case 1:
					Animation1(intensity);
					state = 1;
					if(counter > 200){
						state = 2;
						counter = 0;
					}else{counter = counter +1;}
				break;
				case 2:
					Animation4(intensity);
					if(counter > 200){
						state = 3;
						counter = 0;
					}else{counter = counter +1;}
				break;
				case 3:
					Animation3(intensity);
					state = 3;
					if(counter > 200){
						state = 4;
						counter = 0;
					}else{counter = counter +1;}
```

```
                        break;
                        case 4:
                                RainDrop(Tdata[intensity]);
                                if(counter > 200){
                                        state = 5;
                                        counter = 0;
                                }else{counter = counter +1;}
                                break;
                                case 5:
                                        Animation2(intensity);
                                        if(counter > 200){
                                        state = 6;
                                        counter = 0;
                                }else{counter = counter +1;}
                                break;
                                case 6:
                                        Animation5(intensity);
                                        if(counter > 200){
                                        state = 0;
                                        counter = 0;
                                }else{counter = counter +1;}
                                        break;
                    default:
                                Random();
                                break;
                }
            }
  }
      }
}

//Filename: LEDS_Driver.c
//Author: Duc Tran, Brandon Wong
//Initial Creation Date: November 4, 2013
//Description:
//Lab Number: W 2-3:30
//TA : Omar & Mahesh
//Date of last revision : December 6, 2013
//Hardware Configuration : UTX-2013S304.sch
```

```
#define GPIO_PORTD_DATA_R       (*((volatile unsigned long *)0x400073FC))
#define GPIO_PORTD_DIR_R        (*((volatile unsigned long *)0x40007400))
#define GPIO_PORTD_AFSEL_R      (*((volatile unsigned long *)0x40007420))
#define GPIO_PORTD_DEN_R        (*((volatile unsigned long *)0x4000751C))
#define SYSCTL_RCGC2_GPIOD      0x00000008  // port D Clock Gating Control


//USED PORT A TO TEST IN 1968

#define GPIO_PORTA_DATA_R       (*((volatile unsigned long *)0x400043FC))
#define GPIO_PORTA_DIR_R        (*((volatile unsigned long *)0x40004400))
#define GPIO_PORTA_AFSEL_R      (*((volatile unsigned long *)0x40004420))
#define GPIO_PORTA_DEN_R        (*((volatile unsigned long *)0x4000451C))
#define SYSCTL_RCGC2_GPIOA      0x00000001  // port D Clock Gating Control

#define GPIO_PORTB_DATA_R       (*((volatile unsigned long *)0x400053FC))
#define GPIO_PORTB_DIR_R        (*((volatile unsigned long *)0x40005400))
#define GPIO_PORTB_AFSEL_R      (*((volatile unsigned long *)0x40005420))
#define GPIO_PORTB_DEN_R        (*((volatile unsigned long *)0x4000551C))
#define SYSCTL_RCGC2_GPIOB      0x00000002  // port B Clock Gating Control
#define SYSCTL_RCGC2_R          (*((volatile unsigned long *)0x400FE108))
#define GPIO_PORTB_DR2R_R       (*((volatile unsigned long *)0x40005500))
#define GPIO_PORTB_DR4R_R       (*((volatile unsigned long *)0x40005504))
#define GPIO_PORTB_DR8R_R       (*((volatile unsigned long *)0x40005508))
struct LED{

	unsigned char decoder;
  unsigned char column;		// Output
	unsigned char output;
};
typedef const struct LED LEDType;

LEDType CUBE[25] = {
	{0,0,0x00},
	{0,1,0x01},
	{0,2,0x02},
```

```
	{0,3,0x03},
	{0,4,0x04},
	{0,5,0x05},
	{0,6,0x06},
	{0,7,0x07},

	{1,8,0x08},
	{1,9,0x09},
	{1,10,0x0A},
	{1,11,0x0B},
	{1,12,0x0C},
	{1,13,0x0D},
	{1,14,0x0E},
	{1,15,0x0F},

	{2,16,0x10},
	{2,17,0x11},
	{2,18,0x12},
	{2,19,0x13},
	{2,20,0x14},
	{2,21,0x15},
	{2,22,0x16},
	{2,23,0x17},

	{3,24,0x18}
};

/***************LED_Init***************
 Description: Initializes LED Port
 Input: none
 Output: none
   */
void LED_Init(void){
	volatile unsigned long delay;
	SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOD + SYSCTL_RCGC2_GPIOB; //
activate port D
  delay = SYSCTL_RCGC2_R;
  GPIO_PORTD_DIR_R |= 0xfF;    // make PD5-0 out
  GPIO_PORTD_AFSEL_R &= ~0xfF; // regular port function
```

```
 GPIO_PORTD_DEN_R |= 0xfF;    // enable digital I/O on PD5-0

        GPIO_PORTB_DIR_R |= 0x1F;    // make PB4-0 out
 GPIO_PORTB_AFSEL_R &= ~0x1F; // regular port function
 GPIO_PORTB_DEN_R |= 0x1F;    // enable digital I/O on PB4-0
        GPIO_PORTB_DR2R_R = 0x00;
        GPIO_PORTB_DR8R_R = 0xFF;

}


/***************LED_Tester**************
 Description: Turns on all LEDs
 Input: none
 Output: none
  */
void LED_Tester(void){
        int j;
        unsigned char i;
        while(1){
                for (i = 0; i < 8; i ++){
                        GPIO_PORTD_DATA_R = 0x00;
                        for ( j = 0; j <1000000; j++){}
                }
        }
}

/***************Turn_On**************
 Description: Turns on LED
 Input: int LED number
 Output: none
  */
void Turn_On(int led){
        int index;
        if(led < 25){                   //layer 1
                index = led;
                GPIO_PORTB_DATA_R = 0x01;
                GPIO_PORTD_DATA_R =0x20 + CUBE[index].output;
        }
```

```
	else if(led < 50){		//layer 2
		index =  led -25;
		GPIO_PORTB_DATA_R = 0x02;
		GPIO_PORTD_DATA_R = 0x20 + CUBE[index].output;
	}
	else if(led <75){			//layer 3
		index = led -50;
		GPIO_PORTB_DATA_R = 0x04;
		GPIO_PORTD_DATA_R = 0x20 + CUBE[index].output;
	}
	else if(led < 100){	//layer 4
		index = led - 75;
	  GPIO_PORTB_DATA_R = 0x08;
		GPIO_PORTD_DATA_R = 0x20 + CUBE[index].output;

	}
	else if(led <125){	//layer 5
		index = led -100;
	  GPIO_PORTB_DATA_R = 0x10;
		GPIO_PORTD_DATA_R = 0x20 + CUBE[index].output;
	}

}

/***************Turn_Off**************
 Description: Turns off all LEDs
 Input: none
 Output: none
   */
void Turn_Off(void){
	GPIO_PORTB_DATA_R = 0x00;
}
```