EE445L – Lab 8: Software Drivers for an Embedded System

Michael Park and Jack Zhao
4/8/16

1.0 OBJECTIVE
Requirements document
1. Overview
1.1. Objectives: Why are we doing this project? What is the purpose?
The objectives of this project are to design, build and test an embedded system. Educationally, we are learning how to create a stand-alone system using PCB. It also serves as a comprehensive review of the materials we learned throughout this semester, such as ADC, Speaker, LCD, switch interfacing. Our goal is to create a stand-alone smart display system.

1.2. Roles and Responsibilities: Who will do what? Who are the clients?
The client is our TA Mahesh. Michael and Jack will deisgn the smart display system together. Michael will design the system software and build prototype. Jack will design PCB and write software to pull data from web servers. Together Michael and Jack will integrate the entire system.

1.3. Interactions with Existing Systems: Include this if you are connecting to another board
Our system will be connected to a ESP mini Wifi board.

2. Function Description
2.1. Functionality: What will the system do precisely?
The system is a stand-alone display device used for news updates, weather forcast, temperature, social media notifications, and an alaram clock. More precisely, it will be placed on a desk or a night stand. It will pull data from open servers and display notable information. It will be capable of displaying different time zones and weather zones. It also has an alarm clock functionality. There will be buttons to set alarm and refresh updates on notable information. There will be a light sensor to automatically audjust the brightness of a screen.

2.4. Performance: Define the measures and describe how they will be determined.
The performance will be measured based on the time it takes to retreive data from a server , ADC jitter, and Input/Output current/voltage for speaker.

2.5. Usability: Describe the interfaces. Be quantitative if possible.
Our system will be interfaced with ESP wifi module. There will be four switches interfaced, used for setting time, setting alarm, snoozing alarm, and updating information. It will also be interfaced to an LCD screen and to a speaker. Our speaker will simply be interfaced with a transistor circuit to make buzzing sound for alarm. Lastly, there will be a slidepot used to scroll the screen display sideways. We will use an LCD screen to display time, alarm, weather, calendar, news, stock, gas price, and currency exchange rate. There will be two ISRs. One for switch interface and the other for making sound.

3. Deliverables
3.1. Reports: Simply state the reports for Labs 8 and 11 will be written
Reports for Labs 8 and 11 will be written.
3.2. Outcomes: Simply copy/paste the Lab 8 and Lab 11 deliverables.
Lab8:

A) Objectives
    1-page requirements document
B) Hardware Design
    Regular circuit diagram (SCH file)
C) Software Design
    Include the requirements document
D) Measurement Data
    Time takes to retrieve data from servers
    ADC jitter
    Input/Output voltage, currrent, RMS, of a speaker
E) Analysis and Discussion (none)

Lab11:

A) Objectives
    2-page requirements document
B) Hardware Design
    Detailed circuit diagram of the system (from Lab 7)
C) Software Design (no software printout in the report)
    Briefly explain how your software works (1/2 page maximum)
D) Measurement Data
    Include data as appropriate for your system. Explain how the data was collected.
E) Analysis and Discussion (none). The YouTube video is required

The outcome of the lab8 is different from the prospective outcome of lab11 in a few ways.

For the final project...

- we will pull data from more sources as opposed to only pulling weather data.
- we will change alarm sound to play a rythm/beat
- we will change the screen to scroll sideways
- we will use PCB instead of launchpads
- we will use normal 9V Duracel Battery as a power source
- we will have a case

2.0 HARDWARE DESIGN

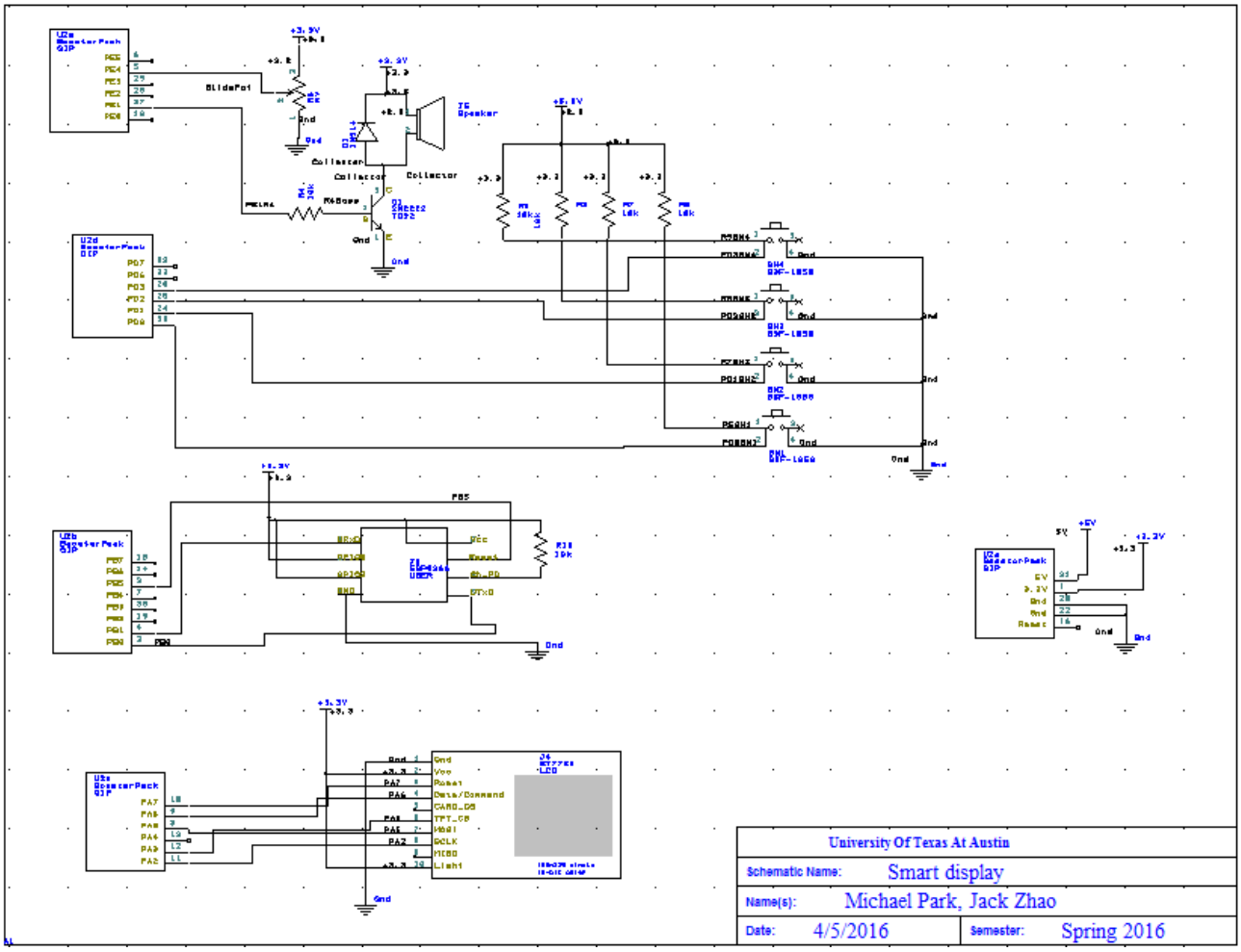

Figure 1: Modified Schematic for Lab8

3.0 SOFTWARE DESIGN

// SetTime.h

```
// Michael Park, Jack Zhao
// Date Created: 02/10/2016
// class for set time function prototypes
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 4/8/16
#include <stdio.h>
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"

enum button_type
{
        DEFAULT,
        INCDEC_TIME_ALARM
};

//global variable for alarm clock
```

```
extern volatile uint16_t time_hours, time_minutes;
extern volatile uint16_t time_flag;

//void changeTime(void);

//IncrementHour
//Increments global variable time_hours
//Input: None
//Output: None
void incrementHour(void);

//DecrementHour
//Decrements global variable time_hours
//Input: None
//Output: None
void decrementHour(void);

//IncrementMin
//Increments global variable time_minutes
//Input: None
//Output: None
void incrementMin(void);

//DeccrementMin
//Deccrements global variable time_minutes
//Input: None
//Output: None
void decrementMin(void);
```

// SetAlarm.h

```
// Michael Park, Jack Zhao
// Date Created: 02/10/2016
// class for functions to set time
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 4/8/16

//global: hour, min, alarmflag, displayflag, switchflag(for interrupt)
#include <stdint.h>

//global variable for alarm clock
extern volatile uint16_t alarm_hours, alarm_minutes;
extern volatile uint16_t alarm_flag;

//IncrementAlarmHour
//Increments global variable alarm_hours
//Input: None
//Output: None
void incrementAlarmHour(void);

//DecrementAlarmHour
//Decrements global variable alarm_hours
```

```
//Input: None
//Output: None
void decrementAlarmHour(void);

//IncrementAlarmMin
//Increments global variable alarm_minutes
//Input: None
//Output: None
void incrementAlarmMin(void);

//DeccrementAlarmMin
//Deccrements global variable alarm_minutes
//Input: None
//Output: None
void decrementAlarmMin(void);
```

// Display.h

```
// Michael Park, Jack Zhao
// Date Created: 2/12/2016
// Includes prototypes of analog and digital time display functions
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 4/6/2016

#include <stdint.h>

#define ACTUALDATE 20

enum display_stat
{
        PG1,
        PG2,
        PG3,
        PG4
};

extern volatile uint16_t display_mode;
extern char weather_temp[3];
extern char weather_weather[6];
extern volatile uint16_t display_status;

//Display
//Builtint Clock
//Builtin Alarm time
//Pulled Weather temperature
//                                        Rainy, Cloudy, Sunny, etc
//
//Recommended Outfit
//                                  100+ : Topless
//                                  80+     : Shorts            +       Short Sleeves
//                                  75+     : Shorts            +       Long Sleeves
//                                  60+     : Pants/Jeans +         Short Sleeves (optional
```

```
jacket)
//                                          50+     : Pants/Jeans +         Long Sleeves
//                                          35+     : Pants/Jeans +  Long Sleeves + Jacket/Hoodie
//                                          35-  : Coat
//SW1  : SetTime
//SW2  : SetAlarm
//SW3  : Snooze
//SW4  : Update Weather Data, Recommneded Outfit will follow
//System Automatically updates at Midnight, Else update switch needs to be pressed to update
void Display_PG1(void);

//Display Calendar
//Display Month and Day In Standard Calendar Grid
//Grid for the Day will be marked red, else black
//Input: SlidePot
void Display_PG2(void);

//Display important news etc
//Input: SlidePot, Update SW4
void Display_PG3(void);

//Display Stock, Currency Exchange rate, Gas Price
//Input: Slidepot, Update SW4
void Display_PG4(void);

// Subroutine to wait 10 msec
// Inputs: None
// Outputs: None
// Notes: ...
void DelayWait10ms(uint32_t n);

//Sunny, Rainy, Cloudy, Stormy, Snowy,
//const char Weather_DB1 [5] = {'S', 'R', 'C', 'S', 'S'};
//const char Weather_DB2 [5] = {'u', 'a', 'l', 't', 'n'};
```

// Switch.h

```
// Runs on LM4F120/TM4C123
// Provide functions that initialize a GPIO as an input pin and
// allow reading of two negative logic switches on PF0 and PF4
// and an external switch on PA5.
// Use bit-banded I/O.
// Daniel and Jonathan Valvano
// September 12, 2013

// negative logic switches connected to PF0 and PF4 on the Launchpad
// red LED connected to PF1 on the Launchpad
// blue LED connected to PF2 on the Launchpad
// green LED connected to PF3 on the Launchpad
// NOTE: The NMI (non-maskable interrupt) is on PF0.  That means that
// the Alternate Function Select, Pull-Up Resistor, Pull-Down Resistor,
// and Digital Enable are all locked for PF0 until a value of 0x4C4F434B
// is written to the Port F GPIO Lock Register.  After Port F is
```

```
// unlocked, bit 0 of the Port F GPIO Commit Register must be set to
// allow access to PF0's control registers.  On the LM4F120, the other
// bits of the Port F GPIO Commit Register are hard-wired to 1, meaning
// that the rest of Port F can always be freely re-configured at any
// time.  Requiring this procedure makes it unlikely to accidentally
// re-configure the JTAG and NMI pins as GPIO, which can lock the
// debugger out of the processor and make it permanently unable to be
// debugged or re-programmed.

#include <stdint.h>

//------------Switch_Init------------
// Initialize GPIO Port A bit 5 for input.
// Input: none
// Output: none
void Switch_Init(void);

//------------Switch_Input------------
// Read and return the status of GPIO Port A bit 5.
// Input: none
// Output: 0x20 if PA5 is high
//         0x00 if PA5 is low
uint32_t Switch_Input(void);

//------------Switch_Debounce------------
// Read and return the status of the switch
// Input: none
// Output: 0x02 if PB1 is high
//         0x00 if PB1 is low
// debounces switch
uint32_t Switch_Debounce(void);

//------------Switch_Debounce------------
// wait for the switch to be touched
// Input: none
// Output: none
// debounces switch
void Switch_WaitForTouch(void);

// ADCSWTrigger.h
// Runs on TM4C123
// Provide functions that initialize ADC0 SS3 to be triggered by
// software and trigger a conversion, wait for it to finish,
// and return the result.
// Daniel Valvano
// August 6, 2015

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
```

```
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: Ain9 (PE4)
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3_Ch9(void);

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: software trigger
// SS3 1st sample source: programmable using variable 'channelNum' [0:7]
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitSWTriggerSeq3(uint32_t channelNum);

// This initialization function sets up the ADC according to the
// following parameters.  Any parameters not explicitly listed
// below are not modified:
// Max sample rate: <=125,000 samples/second
// Sequencer 0 priority: 1st (highest)
// Sequencer 1 priority: 2nd
// Sequencer 2 priority: 3rd
// Sequencer 3 priority: 4th (lowest)
// SS3 triggering event: always trigger
// SS3 1st sample source: programmable using variable 'channelNum' [0:11]
// SS3 interrupts: enabled but not promoted to controller
void ADC0_InitAllTriggerSeq3(uint32_t channelNum);

//------------ADC0_InSeq3------------
// Busy-wait Analog to digital conversion
// Input: none
// Output: 12-bit result of ADC conversion
uint32_t ADC0_InSeq3(void);

//*********************  ESP8266.h  **********************
// Program written by:
// - Steven Prickett  steven.prickett@gmail.com
//
// Brief desicription of program:
// - Initializes an ESP8266 module to act as a WiFi client
//   and fetch weather data from openweathermap.org
//
//*******************************************************
/* Modified by Jonathan Valvano
 Sept 19, 2015
 */
```

```
#ifndef ESP8266_H
#define ESP8266_H

#define ESP8266_ENCRYPT_MODE_OPEN                    0
#define ESP8266_ENCRYPT_MODE_WEP                     1
#define ESP8266_ENCRYPT_MODE_WPA_PSK                 2
#define ESP8266_ENCRYPT_MODE_WPA2_PSK                3
#define ESP8266_ENCRYPT_MODE_WPA_WPA2_PSK  4

#define ESP8266_WIFI_MODE_CLIENT          1
#define ESP8266_WIFI_MODE_AP              2
#define ESP8266_WIFI_MODE_AP_AND_CLIENT   3

//--------------------ESP8266_Init --------------
// initializes the module as a client
// Inputs: none
// Outputs: none
void ESP8266_Init(uint32_t baud);

//-------------------- ESP8266_InitUART--------------------
// intializes uart and gpio needed to communicate with esp8266
// Configure UART1 for serial full duplex operation
// Inputs: baud rate (e.g., 115200 or 9600)
//         echo to UART0?
// Outputs: none
void ESP8266_InitUART(uint32_t baud, int echo);

//---------ESP8266_GetVersionNumber----------
// get status
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_GetVersionNumber(void);

//----------ESP8266_Reset------------
// resets the esp8266 module
// input:  none
// output: 1 if success, 0 if fail
int ESP8266_Reset(void);

//---------ESP8266_SetWifiMode----------
// configures the esp8266 to operate as a wifi client, access point, or both
// Input: mode accepts ESP8266_WIFI_MODE constants
// output: 1 if success, 0 if fail
int ESP8266_SetWifiMode(uint8_t mode);

//---------ESP8266_SetConnectionMux----------
// enables the esp8266 connection mux, required for starting tcp server
// Input: 0 (single) or 1 (multiple)
// output: 1 if success, 0 if fail
int ESP8266_SetConnectionMux(uint8_t enabled);
```

```
//---------ESP8266_CloseTCPConnection----------
// Close TCP connection
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_CloseTCPConnection(void);

//---------ESP8266_DisableServer----------
// disables tcp server
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_DisableServer(void);

//----------ESP8266_JoinAccessPoint------------
// joins a wifi access point using specified ssid and password
// input:  SSID and PASSWORD
// output: 1 if success, 0 if fail
int ESP8266_JoinAccessPoint(const char* ssid, const char* password);

//---------ESP8266_ListAccessPoints----------
// lists available wifi access points
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_ListAccessPoints(void);

//----------ESP8266_ConfigureAccessPoint------------
// configures esp8266 wifi access point settings
// input:  SSID, Password, channel, security
// output: 1 if success, 0 if fail
int ESP8266_ConfigureAccessPoint(const char* ssid, const char* password, uint8_t channel,
uint8_t encryptMode);

//---------ESP8266_GetIPAddress----------
// Get local IP address
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_GetIPAddress(void);

//---------ESP8266_MakeTCPConnection----------
// Establish TCP connection
// Input: IP address or web page as a string
// output: 1 if success, 0 if fail
int ESP8266_MakeTCPConnection(char *IPaddress);

//---------ESP8266_SendTCP----------
// Send a TCP packet to server
// Input: TCP payload to send
// output: 1 if success, 0 if fail
int ESP8266_SendTCP(char* fetch);

//---------ESP8266_SetDataTransmissionMode----------
// set data transmission mode
// Input: 0 not data mode, 1 data mode; return "Link is builded"
```

```
// output: 1 if success, 0 if fail
int ESP8266_SetDataTransmissionMode(uint8_t mode);

//---------ESP8266_GetStatus----------
// get status
// Input: none
// output: 1 if success, 0 if fail
int ESP8266_GetStatus(void);

//--------ESP8266_EnableRXInterrupt--------
// - enables uart rx interrupt
// Inputs: none
// Outputs: none
void ESP8266_EnableRXInterrupt(void);

//--------ESP8266_DisableRXInterrupt--------
// - disables uart rx interrupt
// Inputs: none
// Outputs: none
void ESP8266_DisableRXInterrupt(void);

//--------ESP8266_PrintChar--------
// prints a character to the esp8226 via uart
// Inputs: character to transmit
// Outputs: none
// busy-wait synchronization
void ESP8266_PrintChar(char input);

// ----------ESP8266_QuitAccessPoint-------------
// - disconnects from currently connected wifi access point
// Inputs: none
// Outputs: 1 if success, 0 if fail
int ESP8266_QuitAccessPoint(void);

//**********the following are not tested********
void ESP8266_SetServerTimeout(uint16_t timeout);
void ESP8266_EnableServer(uint16_t port);

// serves a page via the ESP8266
void HTTP_ServePage(const char* body);
#endif
```

// ST7735.h

```
// Runs on LM4F120/TM4C123
// Low level drivers for the ST7735 160x128 LCD based off of
// the file described above.
//    16-bit color, 128 wide by 160 high LCD
// Daniel Valvano, March 30, 2015
// Augmented 7/17/2014 to have a simple graphics facility
// Tested with LaunchPadDLL.dll simulator 9/2/2014

// hardware connections
```

```
// **********ST7735 TFT and SDC******************
// ST7735
// Backlight (pin 10) connected to +3.3 V
// MISO (pin 9) unconnected
// SCK (pin 8) connected to PA2 (SSI0Clk)
// MOSI (pin 7) connected to PA5 (SSI0Tx)
// TFT_CS (pin 6) connected to PA3 (SSI0Fss)
// CARD_CS (pin 5) unconnected
// Data/Command (pin 4) connected to PA6 (GPIO), high for data, low for command
// RESET (pin 3) connected to PA7 (GPIO)
// VCC (pin 2) connected to +3.3 V
// Gnd (pin 1) connected to ground

// **********wide.hk ST7735R with ADXL345 accelerometer ******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - PA2 Sclk SPI clock from microcontroller to TFT or SDC
// !SDA - PA5 MOSI SPI data from microcontroller to TFT or SDC
// DC   - PA6 TFT data/command
// RES  - PA7 TFT reset
// CS   - PA3 TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// SDA  – (NC) I2C data for ADXL345 accelerometer
// SCL  – (NC) I2C clock for ADXL345 accelerometer
// SDO  – (NC) I2C alternate address for ADXL345 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

// **********wide.hk ST7735R with ADXL335 accelerometer ******************
// Silkscreen Label (SDC side up; LCD side down) - Connection
// VCC  - +3.3 V
// GND  - Ground
// !SCL - PA2 Sclk SPI clock from microcontroller to TFT or SDC
// !SDA - PA5 MOSI SPI data from microcontroller to TFT or SDC
// DC   - PA6 TFT data/command
// RES  - PA7 TFT reset
// CS   - PA3 TFT_CS, active low to enable TFT
// *CS  - (NC) SDC_CS, active low to enable SDC
// MISO - (NC) MISO SPI data from SDC to microcontroller
// X– (NC) analog input X-axis from ADXL335 accelerometer
// Y– (NC) analog input Y-axis from ADXL335 accelerometer
// Z– (NC) analog input Z-axis from ADXL335 accelerometer
// Backlight + - Light, backlight connected to +3.3 V

#include <stdint.h>

#ifndef _ST7735H_
#define _ST7735H_

// some flags for ST7735_InitR()
enum initRFlags{
```

```
  none,
  INITR_GREENTAB,
  INITR_REDTAB,
  INITR_BLACKTAB
};

#define ST7735_TFTWIDTH  128
#define ST7735_TFTHEIGHT 160


// Color definitions
#define ST7735_BLACK   0x0000
#define ST7735_BLUE    0xF800
#define ST7735_RED     0x001F
#define ST7735_GREEN   0x07E0
#define ST7735_CYAN    0xFFE0
#define ST7735_MAGENTA 0xF81F
#define ST7735_YELLOW  0x07FF
#define ST7735_WHITE   0xFFFF

//------------ST7735_InitB------------
// Initialization for ST7735B screens.
// Input: none
// Output: none
void ST7735_InitB(void);


//------------ST7735_InitR------------
// Initialization for ST7735R screens (green or red tabs).
// Input: option one of the enumerated options depending on tabs
// Output: none
void ST7735_InitR(enum initRFlags option);


//------------ST7735_DrawPixel------------
// Color the pixel at the given coordinates with the given color.
// Requires 13 bytes of transmission
// Input: x     horizontal position of the pixel, columns from the left edge
//              must be less than 128
//              0 is on the left, 126 is near the right
//        y     vertical position of the pixel, rows from the top edge
//              must be less than 160
//              159 is near the wires, 0 is the side opposite the wires
//        color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawPixel(int16_t x, int16_t y, uint16_t color);

//------------ST7735_DrawFastVLine------------
// Draw a vertical line at the given coordinates with the given height and color.
// A vertical line is parallel to the longer side of the rectangular display
// Requires (11 + 2*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
```

```
//       y     vertical position of the start of the line, rows from the top edge
//       h     vertical height of the line
//       color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastVLine(int16_t x, int16_t y, int16_t h, uint16_t color);


//------------ST7735_DrawFastHLine------------
// Draw a horizontal line at the given coordinates with the given width and color.
// A horizontal line is parallel to the shorter side of the rectangular display
// Requires (11 + 2*w) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the start of the line, columns from the left edge
//       y     vertical position of the start of the line, rows from the top edge
//       w     horizontal width of the line
//       color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_DrawFastHLine(int16_t x, int16_t y, int16_t w, uint16_t color);


//------------ST7735_FillScreen------------
// Fill the screen with the given color.
// Requires 40,971 bytes of transmission
// Input: color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillScreen(uint16_t color);


//------------ST7735_FillRect------------
// Draw a filled rectangle at the given coordinates with the given width, height, and color.
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the top left corner of the rectangle, columns from the left edge
//       y     vertical position of the top left corner of the rectangle, rows from the top edge
//       w     horizontal width of the rectangle
//       h     vertical height of the rectangle
//       color 16-bit color, which can be produced by ST7735_Color565()
// Output: none
void ST7735_FillRect(int16_t x, int16_t y, int16_t w, int16_t h, uint16_t color);


//------------ST7735_Color565------------
// Pass 8-bit (each) R,G,B and get back 16-bit packed color.
// Input: r red value
//       g green value
//       b blue value
// Output: 16-bit color
uint16_t ST7735_Color565(uint8_t r, uint8_t g, uint8_t b);

//------------ST7735_SwapColor------------
// Swaps the red and blue values of the given 16-bit packed color;
// green is unchanged.
// Input: x 16-bit color in format B, G, R
// Output: 16-bit color in format R, G, B
```

```
uint16_t ST7735_SwapColor(uint16_t x) ;


//------------ST7735_DrawBitmap------------
// Displays a 16-bit color BMP image.  A bitmap file that is created
// by a PC image processing program has a header and may be padded
// with dummy columns so the data have four byte alignment.  This
// function assumes that all of that has been stripped out, and the
// array image[] has one 16-bit halfword for each pixel to be
// displayed on the screen (encoded in reverse order, which is
// standard for bitmap files).  An array can be created in this
// format from a 24-bit-per-pixel .bmp file using the associated
// converter program.
// (x,y) is the screen location of the lower left corner of BMP image
// Requires (11 + 2*w*h) bytes of transmission (assuming image fully on screen)
// Input: x     horizontal position of the bottom left corner of the image, columns from the left edge
//        y     vertical position of the bottom left corner of the image, rows from the top edge
//        image pointer to a 16-bit color BMP image
//        w     number of pixels wide
//        h     number of pixels tall
// Output: none
// Must be less than or equal to 128 pixels wide by 160 pixels high
void ST7735_DrawBitmap(int16_t x, int16_t y, const uint16_t *image, int16_t w, int16_t h);

//------------ST7735_DrawCharS------------
// Simple character draw function.  This is the same function from
// Adafruit_GFX.c but adapted for this processor.  However, each call
// to ST7735_DrawPixel() calls setAddrWindow(), which needs to send
// many extra data and commands.  If the background color is the same
// as the text color, no background will be printed, and text can be
// drawn right over existing images without covering them with a box.
// Requires (11 + 2*size*size)*6*8 (image fully on screen; textcolor != bgColor)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
//        textColor 16-bit color of the character
//        bgColor   16-bit color of the background
//        size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2
square)
// Output: none
void ST7735_DrawCharS(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t
size);

//------------ST7735_DrawChar------------
// Advanced character draw function.  This is similar to the function
// from Adafruit_GFX.c but adapted for this processor.  However, this
// function only uses one call to setAddrWindow(), which allows it to
// run at least twice as fast.
// Requires (11 + size*size*6*8) bytes of transmission (assuming image fully on screen)
// Input: x         horizontal position of the top left corner of the character, columns from the left edge
//        y         vertical position of the top left corner of the character, rows from the top edge
//        c         character to be printed
```

```
//       textColor 16-bit color of the character
//       bgColor   16-bit color of the background
//       size      number of pixels per character pixel (e.g. size==2 prints each pixel of font as 2x2
square)
// Output: none
void ST7735_DrawChar(int16_t x, int16_t y, char c, int16_t textColor, int16_t bgColor, uint8_t
size);

//------------ST7735_DrawString------------
// String draw function.
// 16 rows (0 to 15) and 21 characters (0 to 20)
// Requires (11 + size*size*6*8) bytes of transmission for each character
// Input: x         columns from the left edge (0 to 20)
//       y         rows from the top edge (0 to 15)
//       pt        pointer to a null terminated string to be printed
//       textColor 16-bit color of the characters
// bgColor is Black and size is 1
// Output: number of characters printed
uint32_t ST7735_DrawString(uint16_t x, uint16_t y, char *pt, int16_t textColor);;

//********ST7735_SetCursor*****************
// Move the cursor to the desired X- and Y-position.  The
// next character will be printed here.  X=0 is the leftmost
// column.  Y=0 is the top row.
// inputs: newX  new X-position of the cursor (0<=newX<=20)
//         newY  new Y-position of the cursor (0<=newY<=15)
// outputs: none
void ST7735_SetCursor(uint32_t newX, uint32_t newY);

//-----------------------ST7735_OutUDec-----------------------
// Output a 32-bit number in unsigned decimal format
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Input: 32-bit number to be transferred
// Output: none
// Variable format 1-10 digits with no space before or after
void ST7735_OutUDec(uint32_t n);


//------------ST7735_SetRotation------------
// Change the image rotation.
// Requires 2 bytes of transmission
// Input: m new rotation value (0 to 3)
// Output: none
void ST7735_SetRotation(uint8_t m) ;


//------------ST7735_InvertDisplay------------
// Send the command to invert all of the colors.
// Requires 1 byte of transmission
// Input: i 0 to disable inversion; non-zero to enable inversion
// Output: none
```

```
void ST7735_InvertDisplay(int i) ;

// graphics routines
// y coordinates 0 to 31 used for labels and messages
// y coordinates 32 to 159  128 pixels high
// x coordinates 0 to 127   128 pixels wide

// *************** ST7735_PlotClear ********************
// Clear the graphics buffer, set X coordinate to 0
// This routine clears the display
// Inputs: ymin and ymax are range of the plot
// Outputs: none
void ST7735_PlotClear(int32_t ymin, int32_t ymax);

// *************** ST7735_PlotPoint ********************
// Used in the voltage versus time plot, plot one point at y
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotPoint(int32_t y);

// *************** ST7735_PlotLine ********************
// Used in the voltage versus time plot, plot line to new point
// It does output to display
// Inputs: y is the y coordinate of the point plotted
// Outputs: none
void ST7735_PlotLine(int32_t y);

// *************** ST7735_PlotPoints ********************
// Used in the voltage versus time plot, plot two points at y1, y2
// It does output to display
// Inputs: y1 is the y coordinate of the first point plotted
//         y2 is the y coordinate of the second point plotted
// Outputs: none
void ST7735_PlotPoints(int32_t y1,int32_t y2);

// *************** ST7735_PlotBar ********************
// Used in the voltage versus time bar, plot one bar at y
// It does not output to display until RIT128x96x4ShowPlot called
// Inputs: y is the y coordinate of the bar plotted
// Outputs: none
void ST7735_PlotBar(int32_t y);

// *************** ST7735_PlotdBfs ********************
// Used in the amplitude versus frequency plot, plot bar point at y
// 0 to 0.625V scaled on a log plot from min to max
// It does output to display
// Inputs: y is the y ADC value of the bar plotted
// Outputs: none
void ST7735_PlotdBfs(int32_t y);

// *************** ST7735_PlotNext ********************
```

```
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It does not output to display
// Inputs: none
// Outputs: none
void ST7735_PlotNext(void);

// *************** ST7735_PlotNextErase ********************
// Used in all the plots to step the X coordinate one pixel
// X steps from 0 to 127, then back to 0 again
// It clears the vertical space into which the next pixel will be drawn
// Inputs: none
// Outputs: none
void ST7735_PlotNextErase(void);

// Used in all the plots to write buffer to LCD
// Example 1 Voltage versus time
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   ST7735_PlotPoint(data); ST7735_PlotNext(); // called 128 times

// Example 2a Voltage versus time (N data points/pixel, time scale)
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   {   for(j=0;j<N;j++){
//         ST7735_PlotPoint(data[i++]); // called N times
//       }
//       ST7735_PlotNext();
//   }   // called 128 times

// Example 2b Voltage versus time (N data points/pixel, time scale)
//   ST7735_PlotClear(0,4095);  // range from 0 to 4095
//   {   for(j=0;j<N;j++){
//         ST7735_PlotLine(data[i++]); // called N times
//       }
//       ST7735_PlotNext();
//   }   // called 128 times

// Example 3 Voltage versus frequency (512 points)
//   perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//   ST7735_PlotClear(0,1023);  // clip large magnitudes
//   {
//       ST7735_PlotBar(mag[i++]); // called 4 times
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotBar(mag[i++]);
//       ST7735_PlotNext();
//   }   // called 128 times

// Example 4 Voltage versus frequency (512 points), dB scale
//   perform FFT to get 512 magnitudes, mag[i] (0 to 4095)
//   ST7735_PlotClear(0,511);  // parameters ignored
//   {
//       ST7735_PlotdBfs(mag[i++]); // called 4 times
```

```
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotdBfs(mag[i++]);
//       ST7735_PlotNext();
//    }   // called 128 times

// *************** ST7735_OutChar ********************
// Output one character to the LCD
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// Inputs: 8-bit ASCII character
// Outputs: none
void ST7735_OutChar(char ch);

//********ST7735_OutString*****************
// Print a string of characters to the ST7735 LCD.
// Position determined by ST7735_SetCursor command
// Color set by ST7735_SetTextColor
// The string will not automatically wrap.
// inputs: ptr  pointer to NULL-terminated ASCII string
// outputs: none
void ST7735_OutString(char *ptr);

// *************** ST7735_SetTextColor ************************
// Sets the color in which the characters will be printed
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
// ************************************************************
void ST7735_SetTextColor(uint16_t color);

// *************** Output_Init ********************
// Standard device driver initialization function for printf
// Initialize ST7735 LCD
// Inputs: none
// Outputs: none
void Output_Init(void);

// Clear display
void Output_Clear(void);

// Turn off display (low power)
void Output_Off(void);

// Turn on display
void Output_On(void);

// set the color for future output
// Background color is fixed at black
// Input:  16-bit packed color
// Output: none
void Output_Color(uint32_t newColor);
```

```
void ST7735_Line(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2, uint16_t color);

void ST7735_DrawCircle(int16_t x0, int16_t y0, int16_t r, uint16_t color);
#endif
```

// Timer0A.h

```
// Runs on LM4F120/TM4C123
// Use Timer0A in periodic mode to request interrupts at a particular
// period.
// Daniel Valvano
// September 11, 2013

#include <stdint.h>
#ifndef __TIMER0AINTS_H__ // do not include more than once
#define __TIMER0AINTS_H__

extern volatile uint16_t toggleSound;

// ***************** Timer0A_Init ***************
// Activate Timer0A interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in units (1/clockfreq), 32 bits
// Outputs: none
void Timer0A_Init(void(*task)(void), uint32_t period);

#endif // __TIMER0AINTS_H__
```

// Timer1.h

```
// Runs on LM4F120/TM4C123
// Use Timer1 in 32-bit periodic mode to request interrupts at a periodic rate
// Daniel Valvano
// May 5, 2015

#include <stdio.h>
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"


#ifndef __TIMER1INTS_H__ // do not include more than once
#define __TIMER1INTS_H__

extern volatile uint16_t Time_Seconds, Time_Minutes, Time_Hours;
extern volatile uint8_t displayFlag;   //first three bits mean hour, min second

// ***************** Timer1_Init ***************
// Activate Timer1 interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in units (1/clockfreq)
// Outputs: none
void Timer1_Init(void(*task)(void), uint32_t period);
```

```
#endif // __TIMER2INTS_H__
```

```
/*
Alarm Init
*/
void Alarm_Init(void)
{
        //PORTD INIT (For Alarm Sound)
        volatile uint32_t delay;
  SYSCTL_RCGCGPIO_R |= 0x00000010;  // 1) activate clock for Port E
  delay = SYSCTL_RCGCGPIO_R;        // allow time for clock to start

        GPIO_PORTE_CR_R |= 0x02;           // allow changes to PE1
        GPIO_PORTE_DIR_R |= 0x02;        // make PE1 op
  GPIO_PORTE_AFSEL_R &= ~0x02;     // disable alt funct on PE1
  GPIO_PORTE_DEN_R |= 0x02;        // enable digital I/O on PE1
                              // configure PE1 as GPIO
  GPIO_PORTE_PCTL_R = (GPIO_PORTE_PCTL_R&0xFFFFFF0F);
  GPIO_PORTE_AMSEL_R &= ~0x02;
}

// ****************** TIMER1_Init ****************
// Activate TIMER1 interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in units (1/clockfreq)
// Outputs: none
void Timer1_Init(void(*task)(void), uint32_t period){
  SYSCTL_RCGCTIMER_R |= 0x02;   // 0) activate TIMER1
  PeriodicTask = task;          // user function
  TIMER1_CTL_R = 0x00000000;    // 1) disable TIMER1A during setup
  TIMER1_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
  TIMER1_TAMR_R = 0x00000002;   // 3) configure for periodic mode, default down-count
settings
  TIMER1_TAILR_R = period-1;    // 4) reload value
  TIMER1_TAPR_R = 0;            // 5) bus clock resolution
  TIMER1_ICR_R = 0x00000001;    // 6) clear TIMER1A timeout flag
  TIMER1_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI5_R = (NVIC_PRI5_R&0xFFFF00FF)|0x00008000; // 8) priority 4
// interrupts enabled in the main program after all devices initialized
// vector number 37, interrupt number 21
  NVIC_EN0_R = 1<<21;           // 9) enable IRQ 21 in NVIC
  TIMER1_CTL_R = 0x00000001;    // 10) enable TIMER1A
}

#define E1 (*((volatile uint32_t *)0x40024008))
// ****************** Timer0A_Init ****************
// Activate TIMER0 interrupts to run user task periodically
// Inputs:  task is a pointer to a user function
//          period in units (1/clockfreq), 32 bits
// Outputs: none
void Timer0A_Init(void(*task)(void), uint32_t period){long sr;
```

```
  sr = StartCritical();
  SYSCTL_RCGCTIMER_R |= 0x01;   // 0) activate TIMER0
  //PeriodicTask = task;          // user function
  TIMER0_CTL_R = 0x00000000;    // 1) disable TIMER0A during setup
  TIMER0_CFG_R = 0x00000000;    // 2) configure for 32-bit mode
  TIMER0_TAMR_R = 0x00000002;   // 3) configure for periodic mode, default down-count
settings
  TIMER0_TAILR_R = period-1;    // 4) reload value
  TIMER0_TAPR_R = 0;            // 5) bus clock resolution
  TIMER0_ICR_R = 0x00000001;    // 6) clear TIMER0A timeout flag
  TIMER0_IMR_R = 0x00000001;    // 7) arm timeout interrupt
  NVIC_PRI4_R = (NVIC_PRI4_R&0x00FFFFFF)|0x80000000; // 8) priority 4
// interrupts enabled in the main program after all devices initialized
// vector number 35, interrupt number 19
  NVIC_EN0_R = 1<<19;           // 9) enable IRQ 19 in NVIC
  TIMER0_CTL_R = 0x00000001;    // 10) enable TIMER0A
  EndCritical(sr);
}

//Plays Sound Through Speaker
void Timer0A_Handler(void){
  TIMER0_ICR_R = TIMER_ICR_TATOCINT;// acknowledge timer0A timeout
  //(*PeriodicTask)();                // execute user task
          if(toggleSound==1){
                  E1 ^= 0x02;
          }
}

//-------------------ESP8266_Init ----------
// initializes the module as a client
// Inputs: baud rate: tested with 9600 and 115200
// Outputs: none
void ESP8266_Init(uint32_t baud){
  ESP8266_InitUART(baud,true); // baud rate, no echo to UART0
  ESP8266_EnableRXInterrupt();
  SearchLooking = false;
  SearchFound = false;
  ServerResponseSearchLooking = 0; // not looking for "+IPD"
  ServerResponseSearchFinished = 0;
  EnableInterrupts();
// step 1: AT+RST reset module
  printf("ESP8266 Initialization:\n\r");
  ESP8266_EchoResponse = true; // debugging
  if(ESP8266_Reset()==0){
    printf("Reset failure, could not reset\n\r"); while(1){};
  }

    DelayMs(5000);
//  ESP8266SendCommand("AT+UART_CUR=115200,8,1,0,0\r\n");
//  UART_InChar();

//  ESP8266_InitUART(115200,true);
```

```
// step 2: AT+CWMODE=1 set wifi mode to client (not an access point)
  if(ESP8266_SetWifiMode(ESP8266_WIFI_MODE_CLIENT)==0){
    printf("SetWifiMode, could not set mode\n\r"); while(1){};
  }

// step 3: AT+CWJAP="ValvanoAP","12345678"  connect to access point
  if(ESP8266_JoinAccessPoint(SSID_NAME,PASSKEY)==0){
    printf("JoinAccessPoint error, could not join AP\n\r"); while(1){};
  }

// optional step: AT+CIFSR check to see our IP address
  if(ESP8266_GetIPAddress()==0){ // data streamed to UART0, OK
    printf("GetIPAddress error, could not get IP address\n\r"); while(1){};
  }
// optional step: AT+CIPMUX==0 set mode to single socket
  if(ESP8266_SetConnectionMux(0)==0){ // single socket
    printf("SetConnectionMux error, could not set connection mux\n\r"); while(1){};
  }
// optional step: AT+CWLAP check to see other AP in area
  if(ESP8266_ListAccessPoints()==0){
    printf("ListAccessPoints, could not list access points\n\r"); while(1){};
  }
// step 4: AT+CIPMODE=0 set mode to not data mode
  if(ESP8266_SetDataTransmissionMode(0)==0){
    printf("SetDataTransmissionMode, could not make connection\n\r"); while(1){};
  }
  ESP8266_InputProcessingEnabled = false; // not a server
}

//------------ST7735_InitR------------
// Initialization for ST7735R screens (green or red tabs).
// Input: option one of the enumerated options depending on tabs
// Output: none
void ST7735_InitR(enum initRFlags option) {
  commonInit(Rcmd1);
  if(option == INITR_GREENTAB) {
    commandList(Rcmd2green);
    ColStart = 2;
    RowStart = 1;
  } else {
    // colstart, rowstart left at default '0' values
    commandList(Rcmd2red);
  }
  commandList(Rcmd3);

  // if black, change MADCTL color filter
  if (option == INITR_BLACKTAB) {
    writecommand(ST7735_MADCTL);
    writedata(0xC0);
  }
  TabColor = option;
```

```
  ST7735_SetCursor(0,0);
  StTextColor = ST7735_YELLOW;
  ST7735_FillScreen(0);                 // set screen to black
}

//------------Switch_Init------------
// Initialize GPIO Port D
// Input: none
// Output: none
void Switch_Init(void){
  SYSCTL_RCGCGPIO_R |= 0x08;        // 1) activate clock for Port D
          int i;
 // while((SYSCTL_PRGPIO_R&0x02) == 0){};// ready?
  GPIO_PORTD_DIR_R &= ~0x0F;        // PD0-3 is an input
  //GPIO_PORTB_AFSEL_R &= ~0x0F;      // regular port function
  GPIO_PORTD_AMSEL_R &= ~0x0F;      // disable analog on PD0-3
  //GPIO_PORTB_PCTL_R &= ~0x0000FFFF; // PCTL GPIO on PB1
  GPIO_PORTD_DEN_R |= 0x0F;         // PD0-3 enabled as a digital port
          GPIO_PORTD_IS_R &= ~0x0F;                                       // PD 0-3 is edge-
sensitive
          GPIO_PORTD_IBE_R &= ~0x0F;                                  // PD 0-3 is not both edges
          GPIO_PORTD_IEV_R &= ~0x0F;                                  // Pd 0-3 falling edge
event
          GPIO_PORTD_ICR_R = 0x0F;                                        // clear flag 0-3
          GPIO_PORTD_IM_R |= 0x0F;                                        // arm interrupt on
PD 0-3
          //NVIC_PRI0_R = (NVIC_PRI0_R&0xFF00FFFF)|0x00A00000; // (5) priority 5
          NVIC_EN0_R = 0x00000008;                                        //enable interrupt
1(PB) in NVIC
}

//ADC Init
//Initilizes ADC for SlidePot
void ADC0_InitSWTriggerSeq3_Ch9(void){
  SYSCTL_RCGCADC_R |= 0x0001;   // 7) activate ADC0
                        // 1) activate clock for Port E
  SYSCTL_RCGCGPIO_R |= 0x10;
  while((SYSCTL_PRGPIO_R&0x10) != 0x10){};
  GPIO_PORTE_DIR_R &= ~0x10;      // 2) make PE4 input
  GPIO_PORTE_AFSEL_R |= 0x10;     // 3) enable alternate function on PE4
  GPIO_PORTE_DEN_R &= ~0x10;      // 4) disable digital I/O on PE4
  GPIO_PORTE_AMSEL_R |= 0x10;     // 5) enable analog functionality on PE4

//  while((SYSCTL_PRADC_R&0x0001) != 0x0001){};    // good code, but not yet implemented in
simulator


  ADC0_PC_R &= ~0xF;              // 7) clear max sample rate field
  ADC0_PC_R |= 0x1;             //    configure for 125K samples/sec
  ADC0_SSPRI_R = 0x0123;          // 8) Sequencer 3 is highest priority
  ADC0_ACTSS_R &= ~0x0008;        // 9) disable sample sequencer 3
  ADC0_EMUX_R &= ~0xF000;         // 10) seq3 is software trigger
```

```
  ADC0_SSMUX3_R &= ~0x000F;       // 11) clear SS3 field
  ADC0_SSMUX3_R += 9;             //    set channel
  ADC0_SSCTL3_R = 0x0006;         // 12) no TS0 D0, yes IE0 END0
  ADC0_IM_R &= ~0x0008;           // 13) disable SS3 interrupts
  ADC0_ACTSS_R |= 0x0008;         // 14) enable sample sequencer 3
}
```

4.0 MEASUREMENT DATA

1) Time takes to retrieve data

41.71 seconds

2) ADC Jitter

Most frequently sampled ADC value: 1048
Maximum sampled ADC value: 1119
Minimum sampled ADC value: 977
ADC jitter: 142

3)

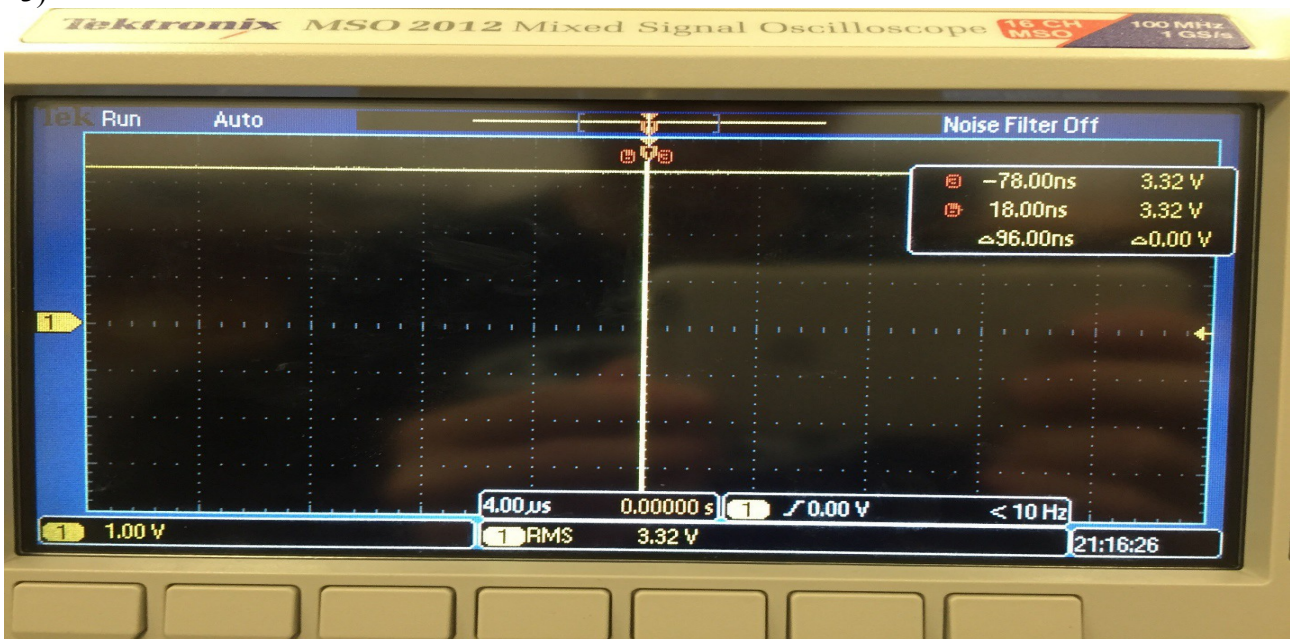

Figure 2: +3.3 V vs time RMS magnitude : 3.32V

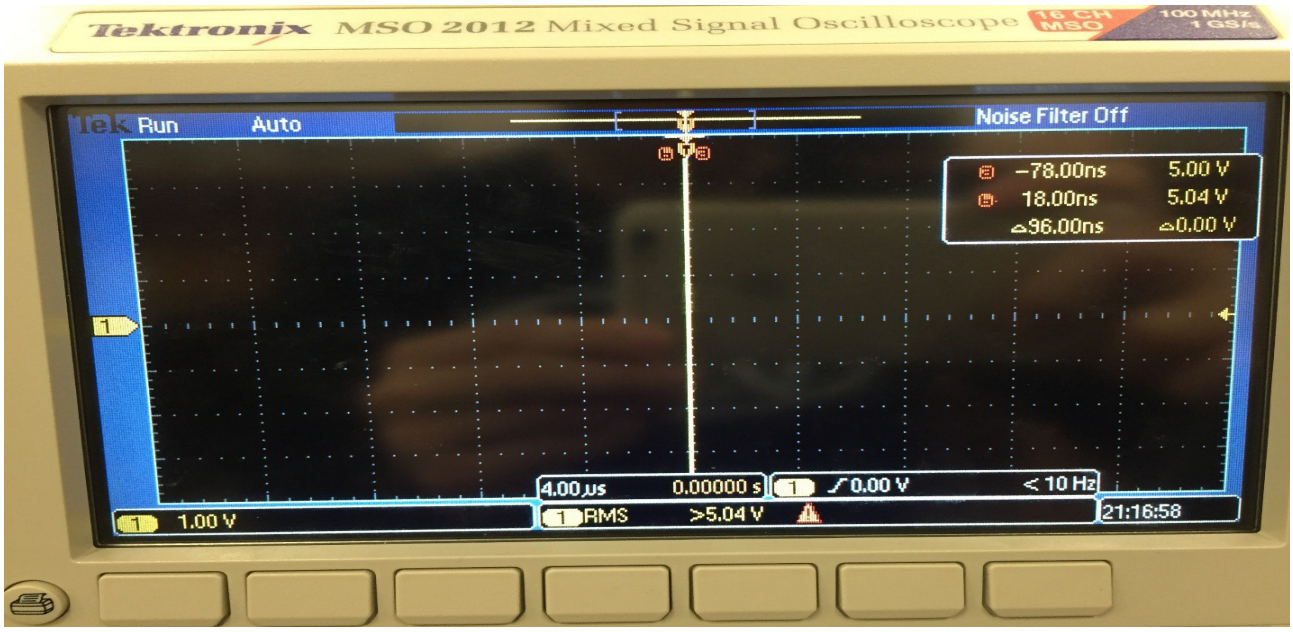

Figure 3: +5.0 V vs time RMS magnitude : 5.04V

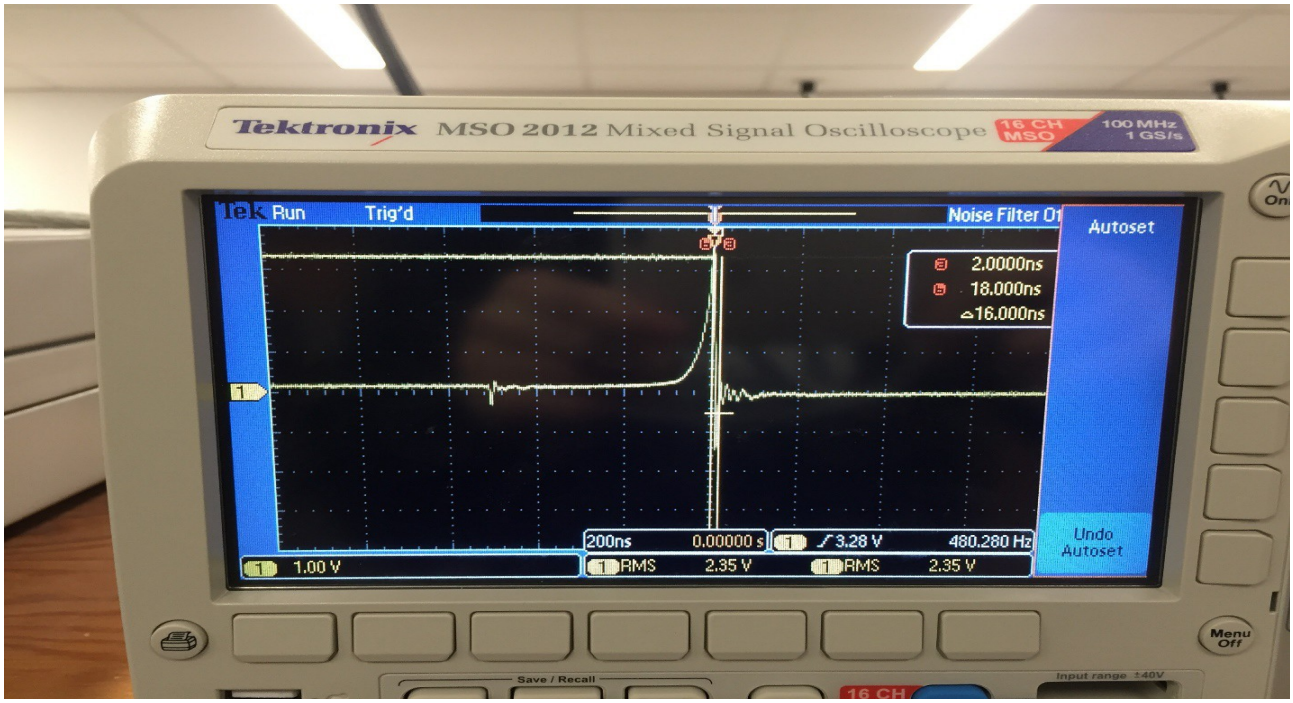

Figure 4: Input Voltage RMS to speaker

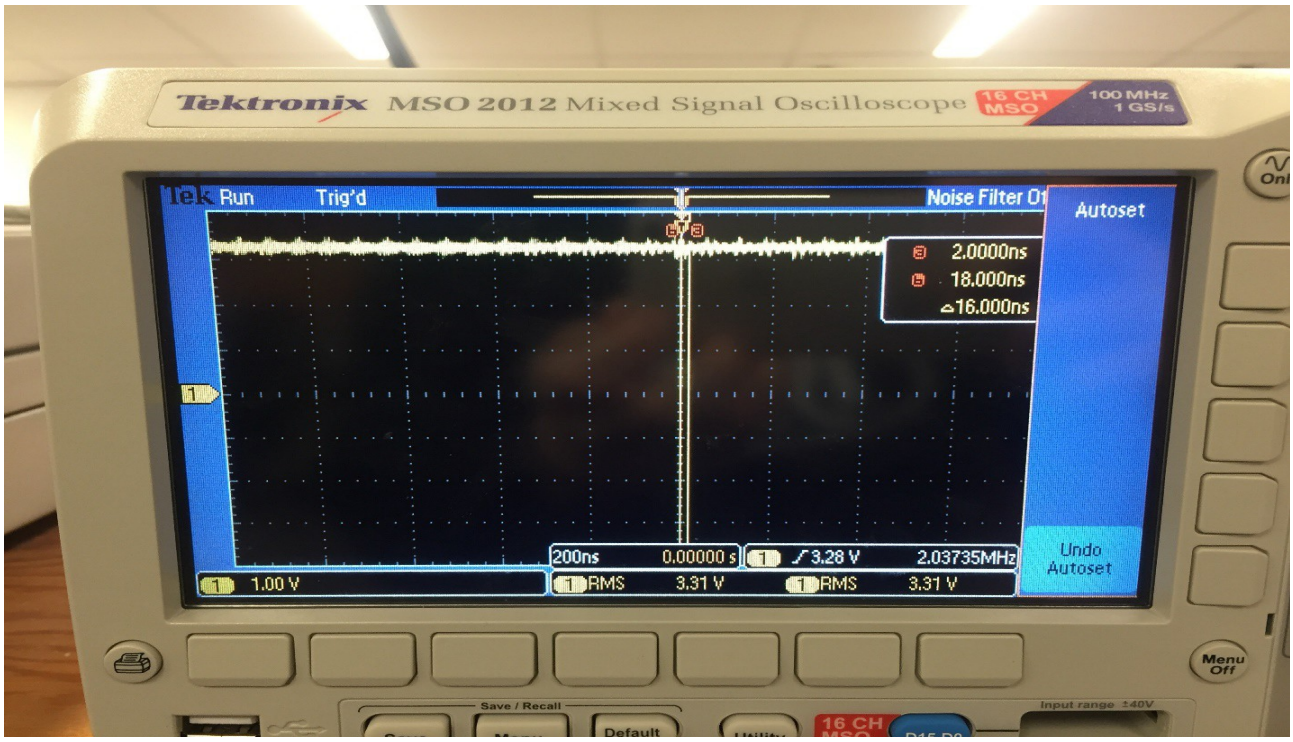

Figure 5: Output Voltage from Speaker

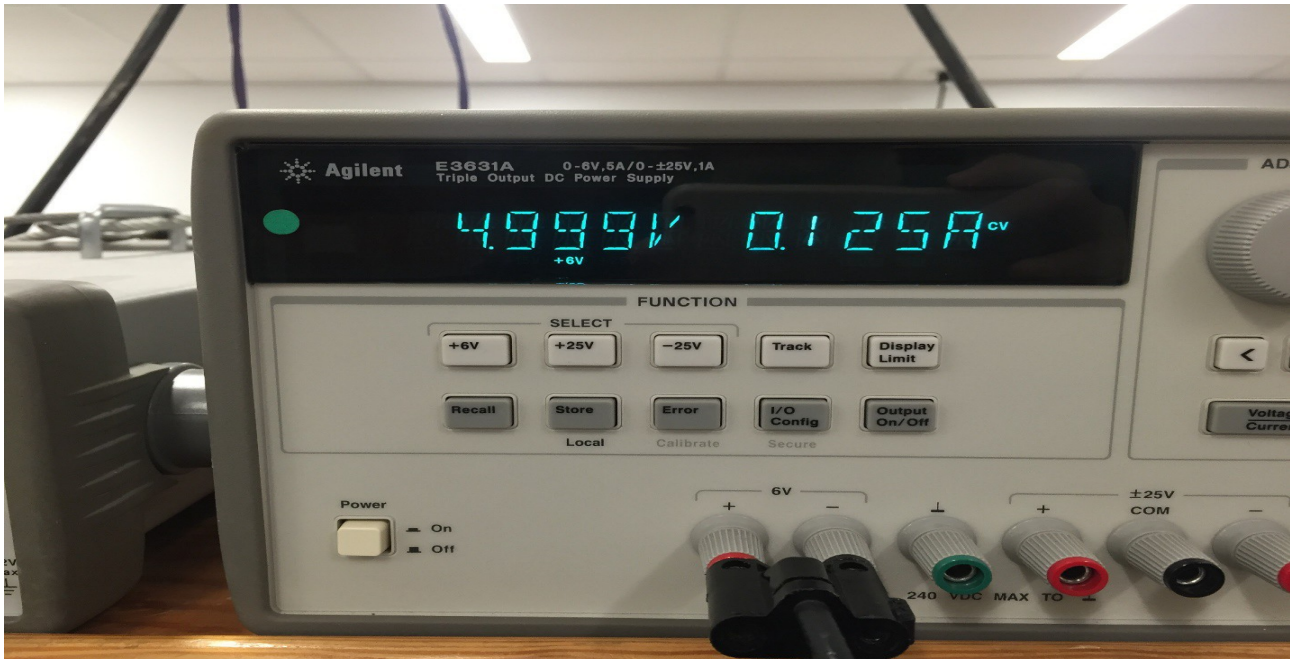

Figure 6: Current Required with Alarm: 0.125A

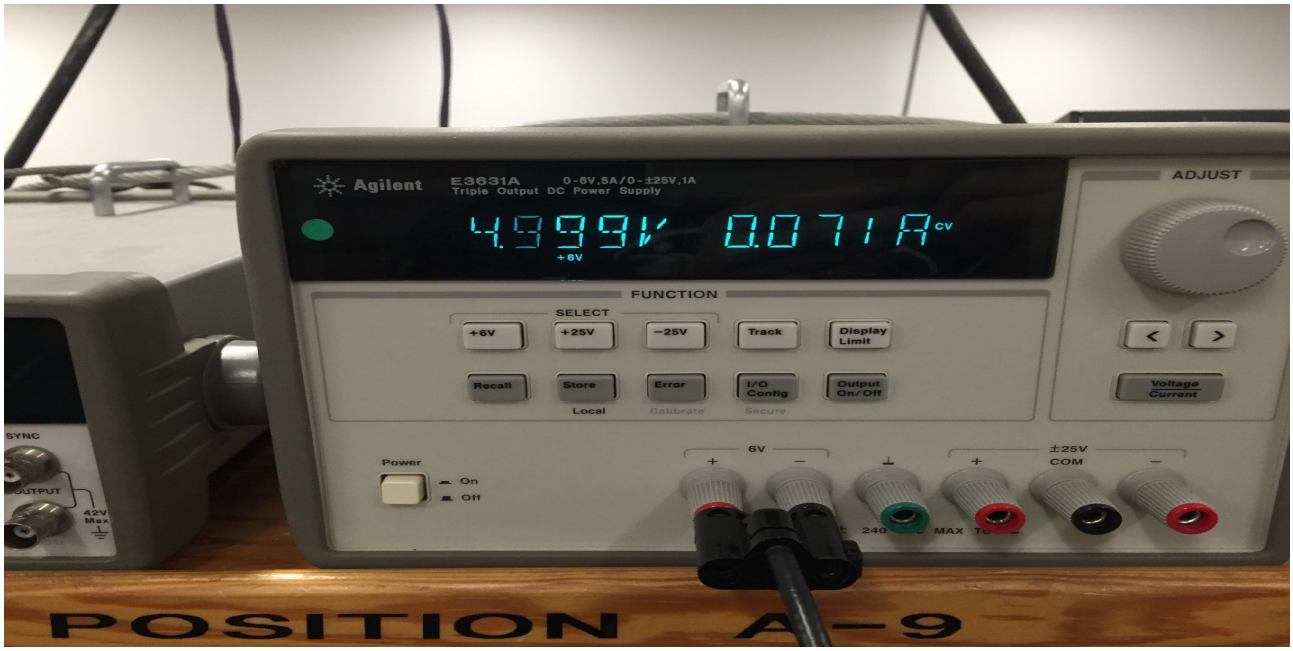

Figure 7: Current Required without Alarm: 0.71A