EE445L – Lab 3: Alarm Clock

Michael Park and Jack Zhao

02/12/16

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 alarm clock. Educationally, we are learning how to design and test modular software and how to perform switch input in the background.

1.2. Process: How will the project be developed?

The project will be developed using the TM4C123 board. The system will be built on a solderless breadboard and run on the usual USB power. The system will use four external switches (B3F-1050), and a speaker. There will be at least four hardware/software modules: switch/keypad input, time management, LCD graphics, and sound output. The process will be to design and test each module independently from the other modules. After each module is tested, the system will be built and tested.

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

Mahesh, our TA is the client. Michael will draw the schematic for the hardware and implement it. Michael will also write a software module for "set time", “set alarm”. Jack will write software modules for "turn on/off alarm" and "display". Michael and Jack will design the framework for the entire system, integrate modules, and test the system together.

1.4. Interactions with Existing Systems: How will it fit in?

The system will use the TM4C123 board, a ST7735 color LCD, a solderless breadboard, and be powered using the USB cable.

1.5. Terminology: Define terms used in the document.

Power budget – using power budget, one can estimate the operation time of a battery-powered embedded system by dividing the energy storage by the average current required to run the system.

Device driver – A collection of software routines that perform I/O functions.

Critical section – Locations within a software module, which if an inerrrupt were to occur at one of these locations, then an error could occur.

Latency -A response time of the computer to external events.

Time jitter – Deviation from the true time of data.

Modular programming – A style of software development that divides the software problem into distinct and independent modules.

1.6. Security: How will intellectual property be managed?

The software written for our projects will be kept in our cloud and hard drive.

2. Function Description

2.1. Functionality: What will the system do precisely?

The clock must be able to perform five functions. 1) It will display hours and minutes in both graphical and numeric forms on the LCD. The graphical output will include the 12 numbers around a circle, the hour hand, and the minute hand. The numerical output will be easy to read. 2) It will allow the operator to set the current time using switches. 3) It will allow the operator to set the alarm time including enabling/disabling alarms. 4) It will make a sound at the alarm time. 5) It will allow the operator to stop the sound. An LED heartbeat will show when the system is running.

2.2. Scope: List the phases and what will be delivered in each phase.

Phase 1 is the preparation; phase 2 is the demonstration; and phase 3 is the lab report. Details can be found in the lab manual.

2.3. Prototypes: How will intermediate progress be demonstrated?

A prototype system running on the TM4C123 board, ST7735 color LCD, and solderless breadboard will be demonstrated. Progress will be judged by the preparation, demonstration and lab report.

2.4. 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 clock display should be beautiful and effective in telling time. Third, the operation of setting the time and alarm should be simple and intuitive. The system should not have critical sections. All shared global variables must be

identified with documentation that a critical section does not exist. Backward jumps in the ISR should be avoided if possible. The interrupt service routine used to maintain time must complete in as short a time as possible. This means all LCD I/O occurs in the main program. The average current on the +5V power will be measured with and without the alarm sounding.

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

There will be four switch inputs. In the main menu, the switches can be used to activate 1) set time; 2) set alarm; 3) turn on/off alarm; and 4) display mode. In set time and alarm modes, two switches add and subtract hours and the other two add and subtract minutes. After 10 seconds of inactivity the system reverts to the main menu. The display mode switch toggles between graphical and numeric displays. The switches will be debounced, so only one action occurs when the operator touches a switch once.

The LCD display shows the time using graphical display typical of a standard on the wall clock. The 12 numbers, the second hand, the minute hand, and the hour hand are large and easy to see. The clock can also display the time in numeric mode using numbers.

The alarm is a simple 440hz square wave. The sound amplitude is just loud enough for the TA to hear when within 3 feet.

2.6. Safety: Explain any safety requirements and how they will be measured.

The alarm sound will be VERY quiet in order to respect other people in the room during testing. Connecting or disconnecting wires on the protoboard while power is applied may damage the board.

3. Deliverables

3.1. Reports: How will the system be described?

A lab report described below is due by 2/19/16. This report includes the final requirements document.

3.2. Audits: How will the clients evaluate progress?

The preparation is due at the beginning of the lab period on 02/11/16.

3.3. Outcomes: What are the deliverables? How do we know when it is done?

There are three deliverables: preparation, demonstration, and report.

2.0 HARDWARE DESIGN

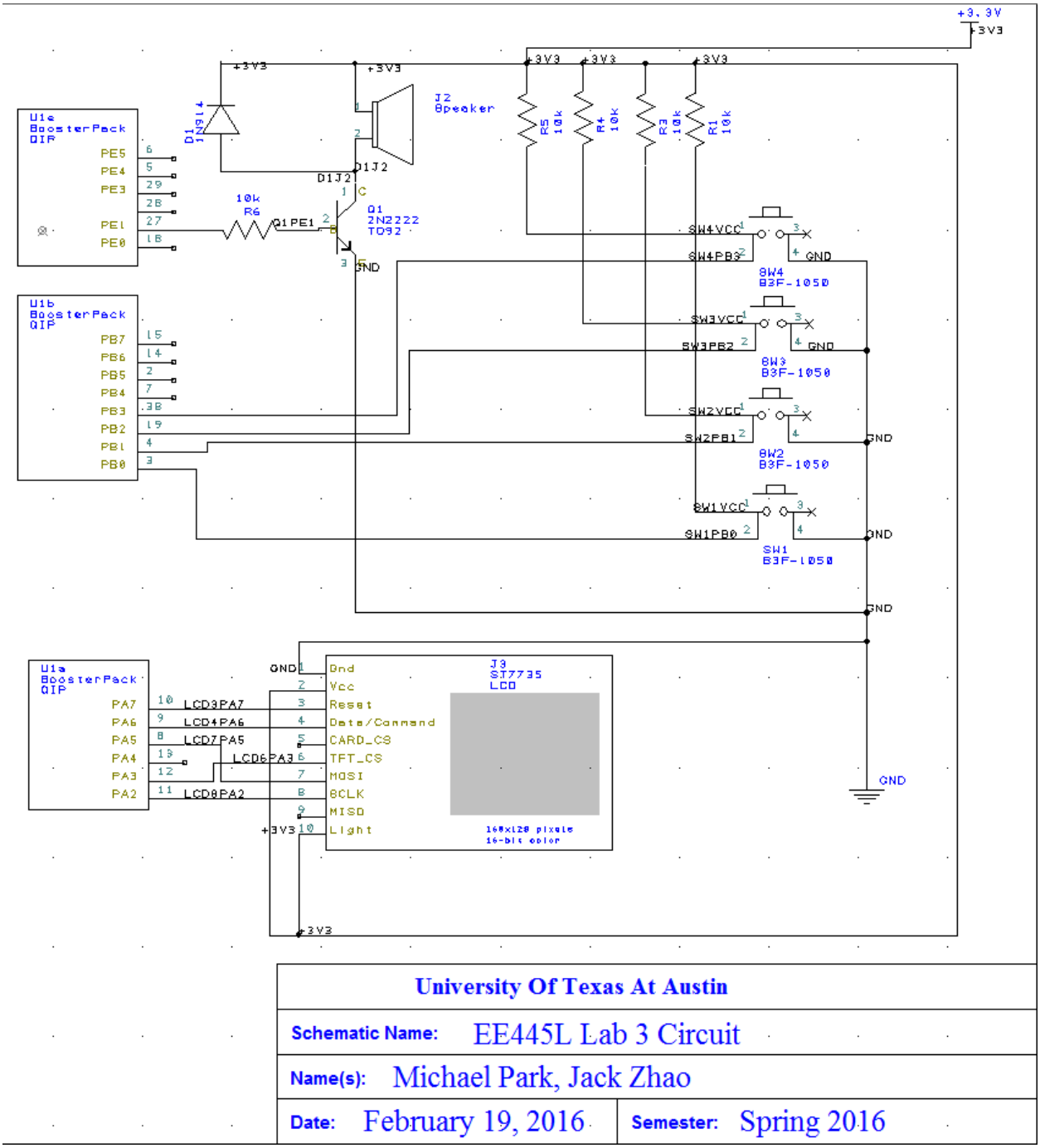

Figure 1: Lab3 Hardware Schematic

3.0 SOFTWARE DESIGN

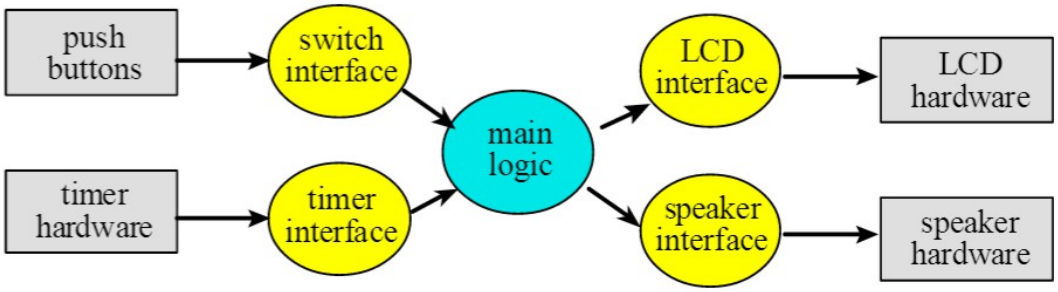

Figure 2: Lab3 Software Data Flow

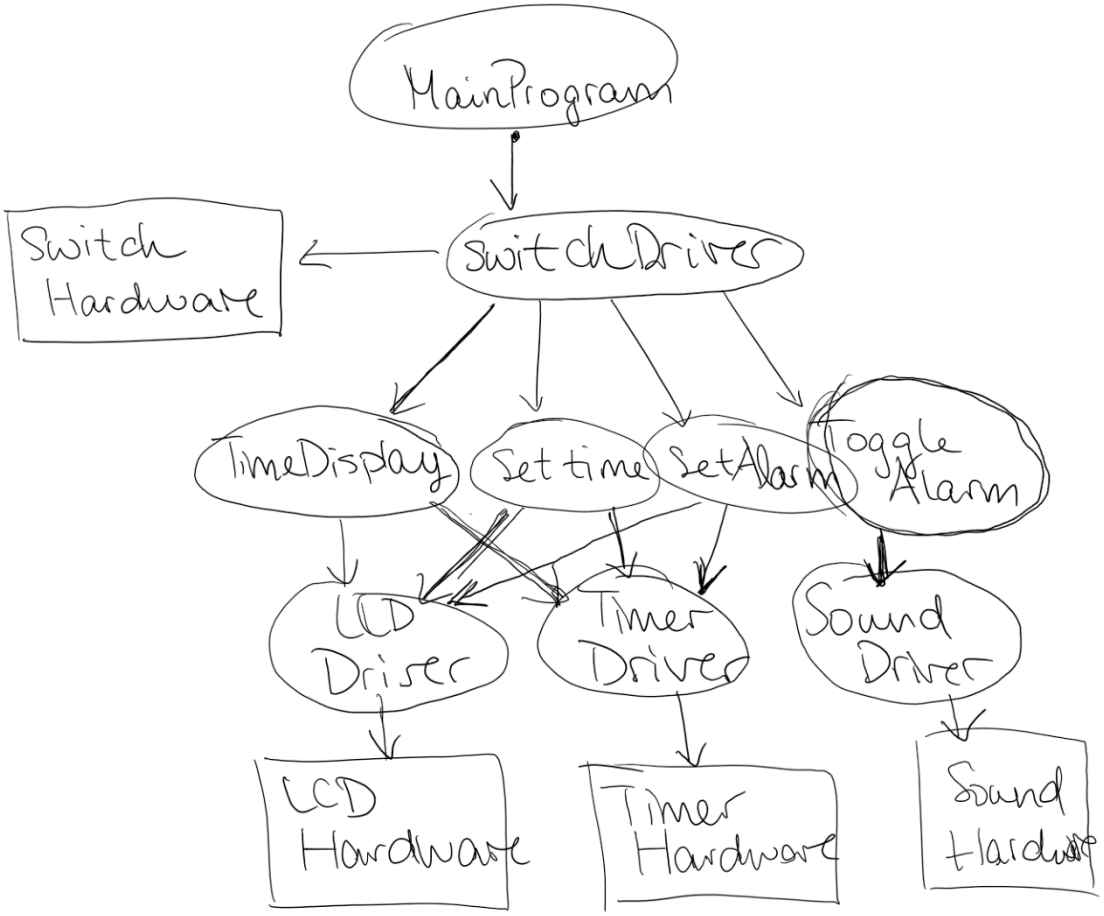

Figure 3: Lab3 Software Call Graph

4.0 MEASUREMENT DATA

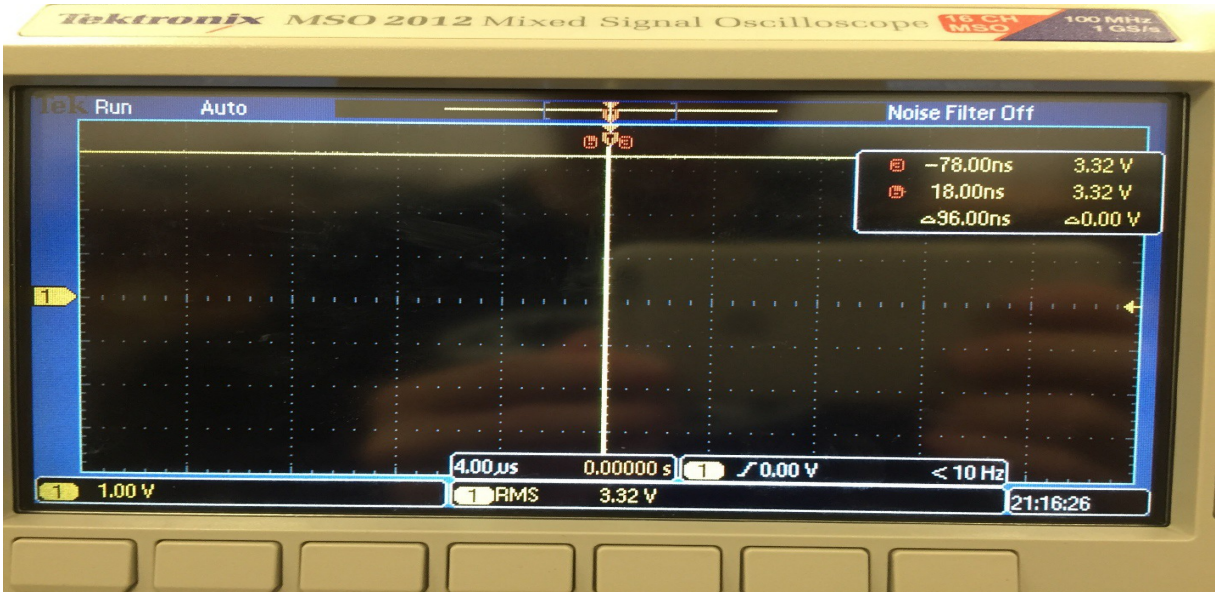

Figure 4: +3.3 V vs time RMS magnitude : 3.32V

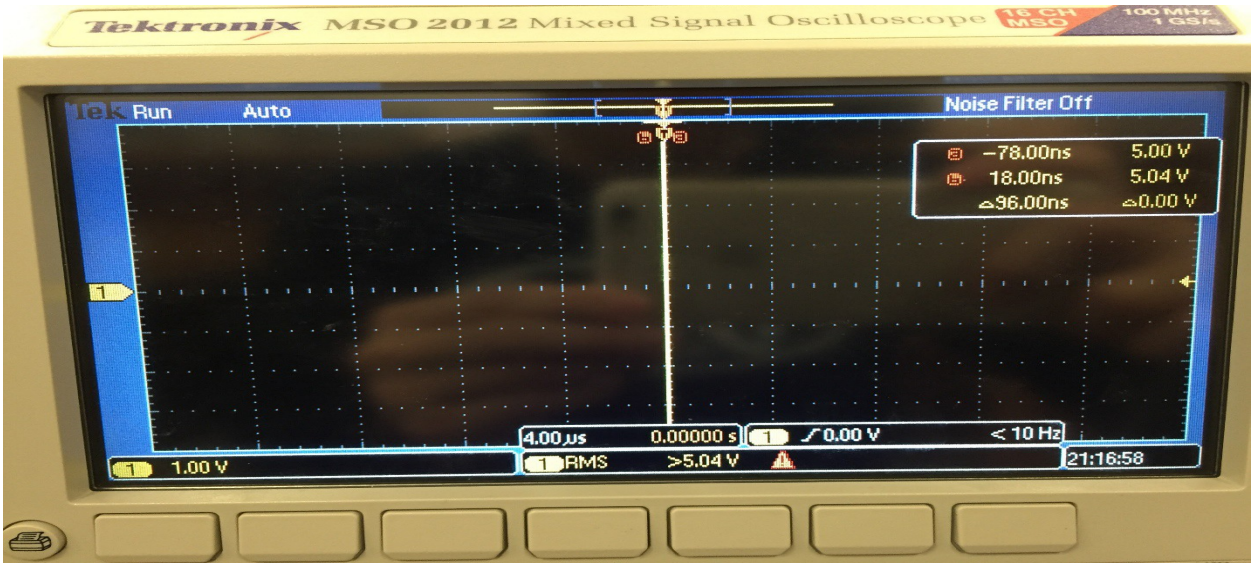

Figure 5: +5.0 V vs time RMS magnitude : 5.04V

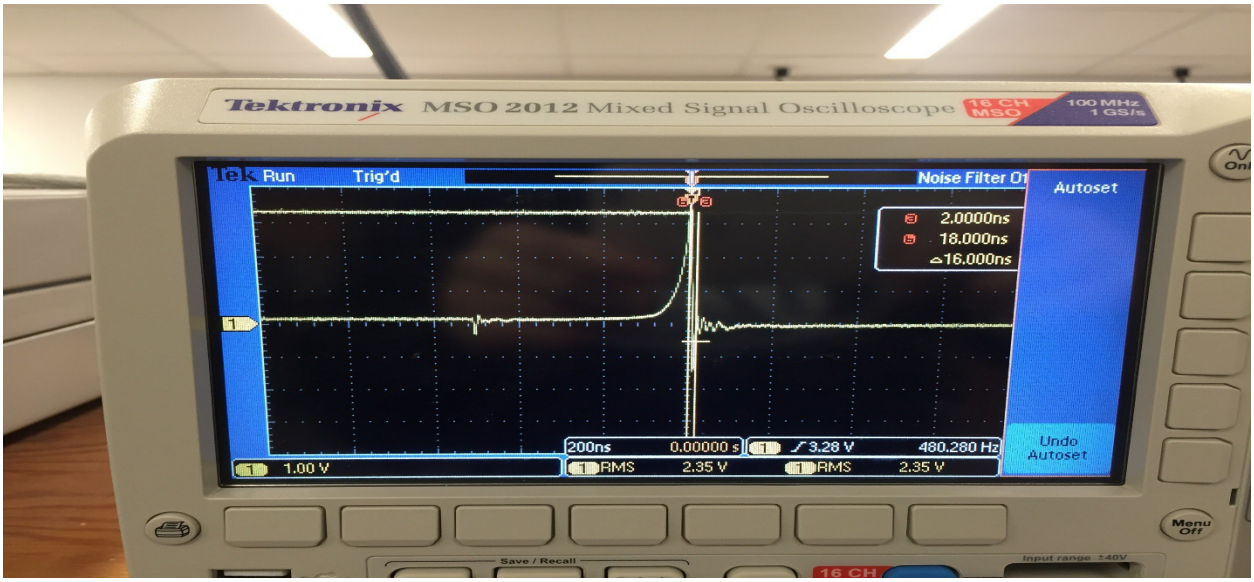

Figure 6: Input Voltage RMS to speaker

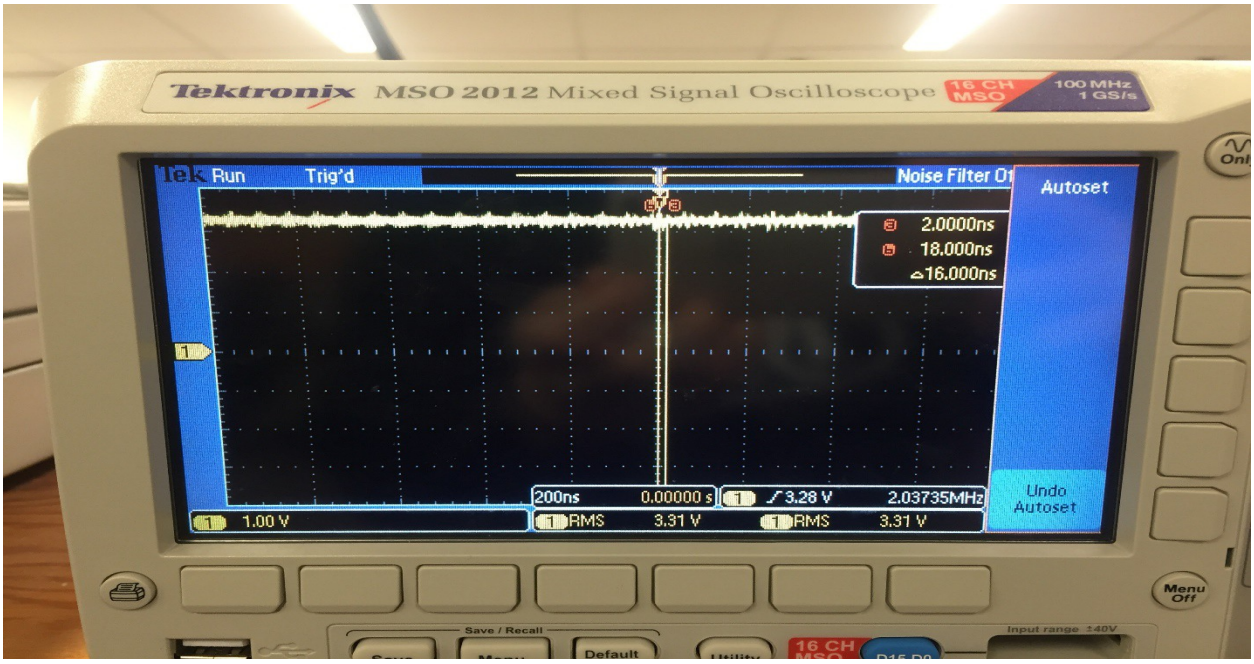

Figure 7: Output Voltage from Speaker

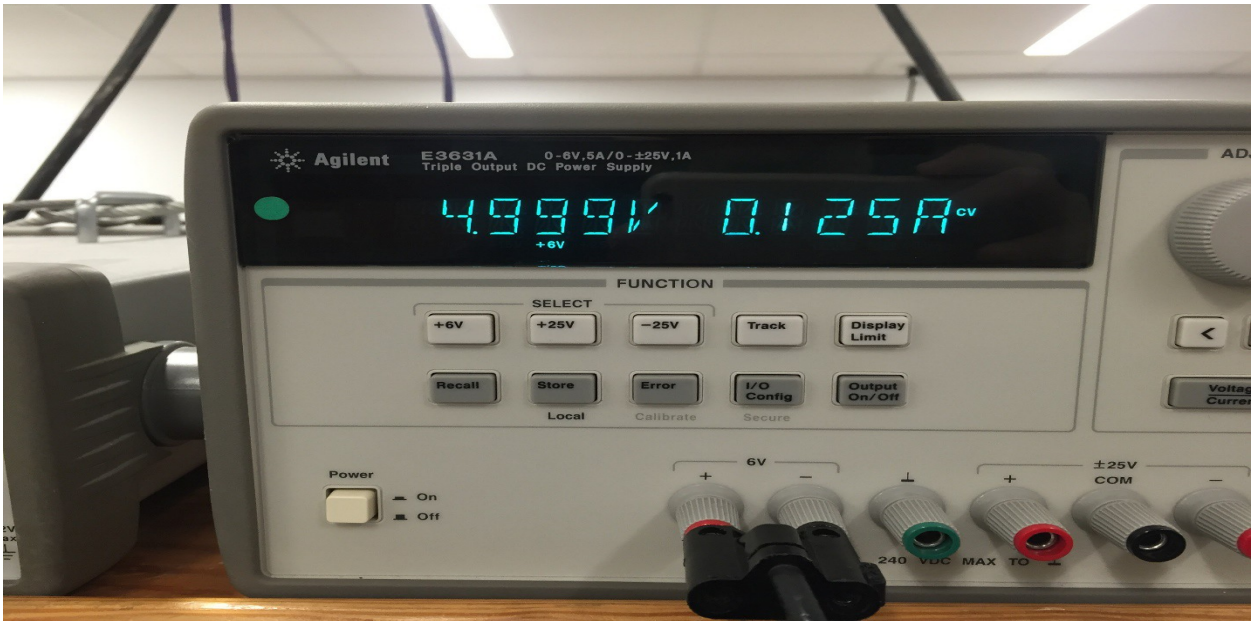

Figure 8: Current Required with Alarm: 0.125A

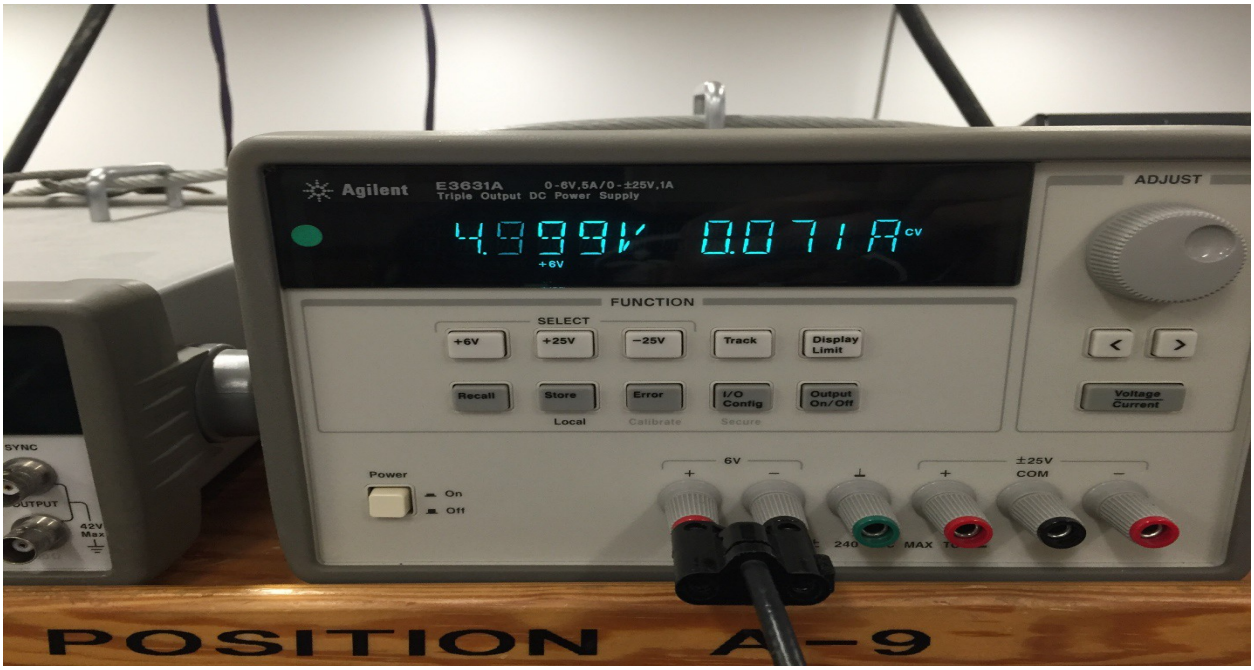

Figure 9: Current Required without Alarm: 0.71A

5.0 ANALYSIS AND DISCUSSION

1) One of the methods to remove critical sections is to, disable the writing of global variables so that no variables are overwritten in execution time. This can be achieved if all we use are local variables and pass in variables as parameters instead of directly writing into a global variable. Another method of removing a critical section is to disable interrupts at the time of writing into global variables. In this situation, no values would be overwritten because the interrupt I will prevent any interrupts to trigger in the critical section.

2) Our first method, digital display, takes approximately 50958 clock cycles to execute the display that updates the LCD with a new time. Our Analog display method for updating the time on the LCD takes approximately 45427 clock cycles to execute.

3) The disadvantage of updating the LCD in the background ISR would be that the program would spend a large chunk of time in the interrupt handler, which is bad programming practice. This is bad because this allows a greater chance of global values being corrupted with overwrite methods. Furthermore, this would cause certain time latencies in real time systems that could be detrimental to the function of the process.

4) We did not redraw the clock for each output. We implemented a method that would “delete” the minute and hour hand after each update. Essentially this method would redraw the previous line, in a black color that would completely overwrite the original hand. In this way we could remove the previous hand and add the new hand after each update, which reduces flicker.

5) The first way we could save power is to use a busy wait system for the main menu instead of using an interrupt, so that we could continuously poll for a button input without the use of multi-threading. Obviously we could reduce sound output or completely eliminate it to reduce battery consumption if it were absolutely necessary. We could implement a system in which the program would go to sleep if the 10 second wait is passed, instead of going to the main menu.

5.0 CODE WRITTEN FOR THIS LAB

```
// AlarmClockMain.c
// Michael Park, Jack Zhao
// Date Created: 2/12/2016
// Driver for Alarm clock. Includes interrupts used
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 2/15/2016

#include <stdio.h>
#include <stdint.h>
#include "ST7735.h"
#include "TimeDisplay.h"
#include "SetTime.h"
#include "PLL.h"
#include "Timer1.h"
#include "Switch.h"
#include "SetAlarm.h"
#include "../Shared/tm4c123gh6pm.h"
#include "PWM.h"
#include "SysTickInts.h"
#include "Common.h"
#include "TimeDisplay.h"
#include "Timer0A.h"

#define PF2       (*((volatile uint32_t *)0x40025010))
#define PF1       (*((volatile uint32_t *)0x40025008))
#define PERIOD                          0x04C4B400
#define Tensec                          0x2FAF0800    //number of cycle for 10s when each
cycle = 12.5ns
```

```
#define A_440 181818



enum ModeSetting
{
	SetTime_Mode,
	SetAlarm_Mode,
	ToggleAlarm_Mode,
	TimeDisplay_Mode
};

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
void MainMenu(void);
void AllowAlarmChangeMode(void);
void DisAllowAlarmChangeMode(void);

volatile uint16_t Mode;
volatile uint16_t active_In10s = 1;
volatile uint32_t counts;
volatile uint32_t timeout=0;
volatile uint16_t toggleSound=1;
volatile uint16_t AllowAlarmChange=1;

int main(void)
{
	//system component setup
```

```
        PLL_Init(Bus80MHz);                 // set system clock to 80 MHz
        Timer1_Init(0, PERIOD);                                         // Init Timer1 for global
clock
 ST7735_InitR(INITR_REDTAB);                                        // Init PORTA and LCD
initializations
        Switch_Init();
                // Init PORTB and switch initialization
        //PWM0A_Init(40000, 30000);                                     // Init PWM
Sound module
        SysTick_Init(80000);


        //PORTE 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;
        Timer0A_Init(0,A_440);



        //PORTF INIT (For Heart Beat)
 SYSCTL_RCGCGPIO_R |= 0x20;  // activate port F
 GPIO_PORTF_DIR_R |= 0x06;   // make PF2 and PF1 output (PF2 built-in LED)
 GPIO_PORTF_AFSEL_R &= ~0x06;// disable alt funct on PF2 and PF1
 GPIO_PORTF_DEN_R |= 0x06;   // enable digital I/O on PF2 and PF1
                        // configure PF2 as GPIO
 GPIO_PORTF_PCTL_R = (GPIO_PORTF_PCTL_R&0xFFFFF00F)+0x00000000;
```

```
 GPIO_PORTF_AMSEL_R = 0;     // disable analog functionality on PF


        //USER FUNCTION INIT                                                    // Self Described Init
Functions
        MainMenu();
        counts = 0;
        Mode = 0xFFFF;
        SysTick_Init(80000);                    // initialize SysTick timer
        EnableInterrupts();                                                   // Enable Interrupts
        //Main Loop
 while(1)
 {
                AllowAlarmChangeMode();
                if((alarm_hours==Time_Hours) && (alarm_minutes==Time_Minutes) && alarm_flag)
                {
                        toggleSound =1;
                        PF1|=0x02;
                }
                else if(alarm_flag)
                {
                        toggleSound =0;
                        PF1 &= 0xFD;
                }
                else if((alarm_hours==Time_Hours) && (alarm_minutes==Time_Minutes)&& (!
alarm_flag))
                {
                        toggleSound =0;
                        PF1 &=0xFD;
                }
                else if(!alarm_flag)
                {
```

```
            toggleSound =0;
            PF1 &=0xFD;
}

if(Mode == SetTime_Mode)
{

            Mode = 0xFFFF; //Acknowledge mode
            active_In10s = 1;
            DisAllowAlarmChangeMode();
            changeTime();
            AllowAlarmChangeMode();
            MainMenu();
}

if(Mode == SetAlarm_Mode)
{
            Mode = 0xFFFF;
            active_In10s = 1;
            DisAllowAlarmChangeMode();
            setAlarmTimeBase();
            AllowAlarmChangeMode();
            MainMenu();
}
if(Mode == ToggleAlarm_Mode)
{
            Mode = 0xFFFF;
            active_In10s = 1;
            MainMenu();
}
```

```
                if(Mode == TimeDisplay_Mode)
                {
                        Mode = 0xFFFF;
                        active_In10s = 1;
                        display_mode = display_mode^0x01;
                        DisAllowAlarmChangeMode();
                        ChooseMode();
                        DisAllowAlarmChangeMode();
                        MainMenu();
                }
 }
}

/*
Initializes to print the Main Menu
*/
void MainMenu(void)
{
        Output_Clear();
        ST7735_DrawString(0, 0, "Set Time", ST7735_YELLOW);
        ST7735_DrawString(0, 3, "Set Alarm", ST7735_YELLOW);
        if (alarm_flag){ST7735_DrawString(0, 6, "Alarm On", ST7735_YELLOW);}
        else {ST7735_DrawString(0, 6, "Alarm Off", ST7735_YELLOW); }
        ST7735_DrawString(0, 9, "Display Mode", ST7735_YELLOW);
}

// Interrupt service routine
// Executed every 12.5ns*(period) = 1ms
void SysTick_Handler(void)
{
```

```
 counts = counts + 1;
        if((counts % 1000) == 0) //every 1s
        {
                PF2 ^= 0x04;            // toggle PF2
                timeout += 1;
        }

        if(timeout == 10)
        {
                active_In10s = 0;
                timeout = 0;
        }
}

/*
GIPOPortB_Handler
ISR for Switch interface: PB0-3
Input: None
Output: None
*/
void GPIOPortB_Handler(void)
{
        //Debouncer
        Switch_Debounce();

        if (GPIO_PORTB_RIS_R & 0X01) //poll PB0
        {
                GPIO_PORTB_ICR_R = 0x01; //acknowledge flag1 and clear
                Mode = SetTime_Mode;
                timeout = 0;
```

```
        }
        if (GPIO_PORTB_RIS_R & 0X02) //poll PB1
        {
                GPIO_PORTB_ICR_R = 0x02; //acknowledge flag1 and clear
                Mode = SetAlarm_Mode;
                timeout = 0;
        }
        if (GPIO_PORTB_RIS_R & 0X04) //poll PB2
        {
                GPIO_PORTB_ICR_R = 0x04; //acknowledge flag1 and clear
                Mode = ToggleAlarm_Mode;
                if(AllowAlarmChange==1){
                alarm_flag ^= 0x1;
                }
                timeout = 0;

        }
        if (GPIO_PORTB_RIS_R & 0X08) //poll PB3
        {
                GPIO_PORTB_ICR_R = 0x08; //acknowledge flag1 and clear
                Mode = TimeDisplay_Mode;
                timeout = 0;
        }
}
void AllowAlarmChangeMode(void){
        AllowAlarmChange=1;
}
void DisAllowAlarmChangeMode(void){
                AllowAlarmChange=0;
}
```

```
// SetTime.c
// Michael Park, Jack Zhao
// Date Created: 02/10/2016
// class for functions to set time
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 02/15/16

#include <stdio.h>
#include <stdint.h>
#include "SetTime.h"
#include "Timer1.h"
#include "TimeDisplay.h"
#include "../Shared/tm4c123gh6pm.h"
#include "Common.h"
#include "ST7735.h"

void changeTime(void)
{
	TIMER1_CTL_R = 0x00000000;    //disable TIMER1A
	Output_Clear();
	DisplaySetTime();

	while (active_In10s)
	{
		switch(Mode)
		{
			case 0:
				Mode = 0xFFFF;
```

```
        incrementHour();
                                        DisplaySetTime();
                                        DelayWait10ms(1);
                                        break;
      case 1:
                                        Mode = 0xFFFF;
        decrementHour();
                                        DisplaySetTime();
        DelayWait10ms(1);
                                        break;
      case 2:
                                        Mode = 0xFFFF;
                                        incrementMin();
                                        DisplaySetTime();
                                        DelayWait10ms(1);
        break;
      case 3:
                                        Mode = 0xFFFF;
                                        decrementMin();
                                        DisplaySetTime();
                                        DelayWait10ms(1);
        break;
                              default:
                                        Mode = 0xFFFF;
                                        break;
                    }
            }
            TIMER1_CTL_R = 0x00000001;    //enable TIMER1A
}
```

```
void incrementHour(void)
{
  if((Time_Hours+1)>23) {Time_Hours = 0;} //if hour exceeds 23, reset to 0
  else {Time_Hours++;}
}

void decrementHour(void)
{
  if((Time_Hours-1)<0) {Time_Hours = 23;} //if hour below 0, reset to 23
  else {Time_Hours--;}
}

void incrementMin(void)
{
  if((Time_Minutes+1)>59) {Time_Minutes = 0;} //if min exceeds 59, reset to 0
  else {Time_Minutes++;}
}

void decrementMin(void)
{
  if((Time_Minutes-1)<0) {Time_Minutes = 59;} //if min below 0, reset to 59
  else {Time_Minutes--;}
}
// SetTime.c
// Michael Park, Jack Zhao
// Date Created: 02/10/2016
// class for functions to set alarm
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 02/15/16
```

```
#include <stdio.h>
#include <stdint.h>
#include "SetTime.h"
#include "Timer1.h"
#include "TimeDisplay.h"
#include "../Shared/tm4c123gh6pm.h"
#include "Common.h"
#include "ST7735.h"
#include "SetAlarm.h"

volatile uint16_t alarm_hours=0;
volatile uint16_t alarm_minutes=0;
volatile uint16_t alarm_flag = 1;


//Assuming Time interrupt is disalbed
void setAlarmTimeBase(void)
{
	Output_Clear();
	DisplayAlarm();

	while (active_In10s)
	{
		switch(Mode)
		{
			case 0:
				Mode = 0xFFFF;
      incrementAlarmHour();
				DisplayAlarm();
```

```
                                    DelayWait10ms(1);
                                    break;
    case 1:
                                    Mode = 0xFFFF;
     decrementAlarmHour();
                                    DisplayAlarm();
     DelayWait10ms(1);
                                    break;
    case 2:
                                    Mode = 0xFFFF;
                                    incrementAlarmMin();
                                    DisplayAlarm();
                                    DelayWait10ms(1);
     break;
    case 3:
                                    Mode = 0xFFFF;
                                    decrementAlarmMin();
                                    DisplayAlarm();
                                    DelayWait10ms(1);
     break;
                            default:
                                    Mode = 0xFFFF;
                                    break;
                    }
          }

}

void incrementAlarmHour(void)
{
```

```
 if((alarm_hours+1)>23) {alarm_hours = 0;} //if hour exceeds 23, reset to 0
 else {alarm_hours++;}
}

void decrementAlarmHour(void)
{
 if((alarm_hours-1)<0) {alarm_hours = 23;} //if hour below 0, reset to 23
 else {alarm_hours--;}
}

void incrementAlarmMin(void)
{
 if((alarm_minutes+1)>59) {alarm_minutes = 0;} //if min exceeds 59, reset to 0
 else {alarm_minutes++;}
}

void decrementAlarmMin(void)
{
 if((alarm_minutes-1)<0) {alarm_minutes = 59;} //if min below 0, reset to 59
 else {alarm_minutes--;}
}
void AlarmToggleON(void){

                if(alarm_flag==0)
                        alarm_flag =1;

}
void AlarmToggleOFF(void){
                        if(alarm_flag==1)
                        alarm_flag =0;
```

```
}
// ATimeDisplay.c
// Michael Park, Jack Zhao
// Date Created: 2/12/2016
// Includes analog and digital time display functions
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 2/15/2016

#include <stdint.h>
#include <stdio.h>
#include "../Shared/tm4c123gh6pm.h"
#include "ST7735.h"
#include <math.h>
#include "TimeDisplay.h"
#include "Timer1.h"
#include "SetAlarm.h"
#include "Common.h"
#include "Systick.h"
#include "Stopwatch.h"

#define CENTER_X 64
#define CENTER_Y 80
#define LENGTH_S 45.0
#define LENGTH_M 45.0
#define LENGTH_H 45.0
#define PI 3.14159265358979323846
```

```
extern long StartCritical (void);    // previous I bit, disable interrupts
extern void EndCritical(long sr);    // restore I bit to previous value
void DelayWait10ms(uint32_t n);
void EraseSecond(void);

//extern volatile uint16_t Time_Minutes, Time_Hours;
volatile uint16_t display_mode = 0;
//volatile double x, y;
volatile static double prevsx, prevsy, prevmx, prevmy, prevhx, prevhy;
extern volatile uint16_t Stopwatch_Triggered;
extern volatile uint16_t alarm_minutes, alarm_hours;
extern volatile uint16_t Time_minutes, Time_hours;
extern volatile uint16_t alarm_flag;
void DisplayAlarm(void)
{
        char AsciiArray[] = {'0','1','2','3','4','5','6','7','8','9'};
        char ch[5];
        ch[0] = AsciiArray[alarm_hours/10];
        ch[1] = AsciiArray[alarm_hours%10];
        ch[2] = ':';
        ch[3] = AsciiArray[alarm_minutes/10];
        ch[4] = AsciiArray[alarm_minutes%10];

        Output_Clear();
        ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 4);
        ST7735_DrawCharS(25, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 4);
        ST7735_DrawCharS(50, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 4);
        ST7735_DrawCharS(75, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 4);
        ST7735_DrawCharS(100, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 4);
}
```

```
void DisplaySetTime(void)
{
	char AsciiArray[] = {'0','1','2','3','4','5','6','7','8','9'};
	char ch[5];
	ch[0] = AsciiArray[Time_Hours/10];
	ch[1] = AsciiArray[Time_Hours%10];
	ch[2] = ':';
	ch[3] = AsciiArray[Time_Minutes/10];
	ch[4] = AsciiArray[Time_Minutes%10];

	Output_Clear();
	ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 4);
	ST7735_DrawCharS(25, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 4);
	ST7735_DrawCharS(50, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 4);
	ST7735_DrawCharS(75, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 4);
	ST7735_DrawCharS(100, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 4);
}

void DisplaySecond(void)
{
	volatile static double x, y;
	x = 0;
	y = 0;
	x = CENTER_X + LENGTH_S*cos(Time_Seconds*6*(PI/180)-1.58);
	y = CENTER_Y + LENGTH_S*sin(Time_Seconds*6*(PI/180)-1.58);
	prevsx = x;
	prevsy = y;

	ST7735_Line(CENTER_X, CENTER_Y, x, y, ST7735_WHITE);
```

```
}

void EraseSecond(void)
{
	if (((prevsx==prevmx)&&(prevsy==prevmy)) || ((prevsx==prevmx)&&(prevsy==prevmy))){}
	else{ ST7735_Line(CENTER_X, CENTER_Y, prevsx, prevsy, ST7735_BLACK);}
}
void DisplayMinute(void)
{
	volatile static double x, y;
	x = 0;
	y = 0;
	x = CENTER_X + LENGTH_M*cos(Time_Minutes*6*(PI/180)-1.58);
	y = CENTER_Y + LENGTH_M*sin(Time_Minutes*6*(PI/180)-1.58);
	prevmx = x;
	prevmy = y;

	ST7735_Line(CENTER_X, CENTER_Y, x, y, ST7735_WHITE);
}

void EraseMinute(void)
{
	if (((prevmx==prevhx)&&(prevmy==prevhy))){}
	else{ ST7735_Line(CENTER_X, CENTER_Y, prevmx, prevmy, ST7735_BLACK);}
}

void DisplayHour(void)
{
	volatile static double x, y;
	x = 0;
```

```
        y = 0;
        x = CENTER_X + LENGTH_H*cos(Time_Hours*30*(PI/180)-1.58);
        y = CENTER_Y + LENGTH_H*sin(Time_Hours*30*(PI/180)-1.58);
        prevhx = x;
        prevhy = y;

        ST7735_Line(CENTER_X, CENTER_Y, x, y, ST7735_WHITE);
}

void EraseHour(void)
{
        ST7735_Line(CENTER_X, CENTER_Y, prevhx, prevhy, ST7735_BLACK);
}

void ChooseMode(void)
{
        if (display_mode==1) {DisplayAnalog();}
        else if(display_mode==0) {DisplayDigital();}
        else {}
}

void DisplayAnalog(void)
{int32_t timeDisplay;
        timeDisplay=NVIC_ST_CURRENT_R;
                Output_Clear();
                ST7735_SetCursor(0, 0);
                ST7735_OutString("alarm:");
                ST7735_OutUDec(alarm_hours);
                ST7735_OutString(":");
                ST7735_OutUDec(alarm_minutes);
```

```
ClockFace_Init();
DisplaySecond();
DisplayMinute();
DisplayHour();
timeDisplay=timeDisplay-NVIC_ST_CURRENT_R;
while(active_In10s)
{
        timeDisplay=NVIC_ST_CURRENT_R;
        if(Mode == 3)
        {
                break;
        }
        if(displayFlag == 0x01)
        {
                displayFlag = 0xFF;
                EraseSecond();
                DisplaySecond();
        }
        if(displayFlag == 0x03)
        {
                displayFlag = 0xFF;
                EraseSecond();
                DisplaySecond();
                EraseMinute();
                DisplayMinute();
        }
        if(displayFlag == 0x07)
        {
                displayFlag = 0xFF;
```

```
                    EraseSecond();
                    DisplaySecond();
                    EraseMinute();
                    DisplayMinute();
                    EraseHour();
                    DisplayHour();
                }
                timeDisplay=NVIC_ST_CURRENT_R-timeDisplay;
                timeDisplay++;
            }
}

void DisplayDigital(void)
{//int32_t timeDisplay;
    //timeDisplay=NVIC_ST_CURRENT_R;
    char AsciiArray[] = {'0','1','2','3','4','5','6','7','8','9'};
    char ch[5];

    ch[0] = AsciiArray[Time_Hours/10];
    ch[1] = AsciiArray[Time_Hours%10];
    ch[2] = ':';
    ch[3] = AsciiArray[Time_Minutes/10];
    ch[4] = AsciiArray[Time_Minutes%10];

    Output_Clear();
    ST7735_SetCursor(0, 0);
    ST7735_OutString("alarm:");
    ST7735_OutUDec(alarm_hours);
    ST7735_OutString(":");
    ST7735_OutUDec(alarm_minutes);
```

```
ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 4);
ST7735_DrawCharS(25, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 4);
ST7735_DrawCharS(50, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 4);
ST7735_DrawCharS(75, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 4);
ST7735_DrawCharS(100, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 4);
//timeDisplay=timeDisplay-NVIC_ST_CURRENT_R;

while(active_In10s)
{
	//timeDisplay=NVIC_ST_CURRENT_R;
	if(Mode == 3)
	{
		break;
	}
	if(Mode==0){
		if((Time_Minutes+5)<60){
			alarm_minutes=Time_Minutes+5;
		alarm_hours=Time_Hours;
			alarm_flag=1;
		}

	}

		ch[0] = AsciiArray[Time_Hours/10];
		ch[1] = AsciiArray[Time_Hours%10];
		ch[2] = ':';
		ch[3] = AsciiArray[Time_Minutes/10];
		ch[4] = AsciiArray[Time_Minutes%10];
```

```
                        if(displayFlag == 0x03)
                        {
                                displayFlag = 0xFF;
                                Output_Clear();
                                ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(25, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(50, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(75, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(100, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 4);
                        }
                        if(displayFlag == 0x07)
                        {
                                displayFlag = 0xFF;
                                Output_Clear();
                                ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(25, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(50, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(75, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 4);
                                ST7735_DrawCharS(100, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 4);
                        }
                        //timeDisplay=timeDisplay-NVIC_ST_CURRENT_R;
                        //timeDisplay++;
                }
}

void ClockFace_Init(void)
{
   uint32_t r = 60;
   ST7735_DrawCircle(64, 80, r, ST7735_Color565(0, 0, 255));  // clock face
                ST7735_DrawCharS(57, 22, '1', ST7735_Color565(255, 0, 0), 0, 1);
```

```
   ST7735_DrawCharS(62, 22, '2', ST7735_Color565(255, 0, 0), 0, 1);
                    ST7735_DrawCharS(117, 77, '3', ST7735_Color565(255, 128, 0), 0, 1);
   ST7735_DrawCharS(61, 132, '6', ST7735_Color565(255, 0, 0), 0, 1);
   ST7735_DrawCharS(6, 77, '9', ST7735_Color565(255, 0, 0), 0, 1);
   DelayWait10ms(1);
}

// Subroutine to wait 10 msec
// Inputs: None
// Outputs: None
// Notes: ...
void DelayWait10ms(uint32_t n)
{
        uint32_t volatile time;
  while(n){
   time = 727240*2/91;  // 10msec
   while(time){
                    time--;
   }
   n--;
  }
}
//------------Switch_Init------------
// Initialize GPIO Port A bit 5 for input
// Input: none
// Output: none
void Switch_Init(void){
  SYSCTL_RCGCGPIO_R |= 0x02;       // 1) activate clock for Port B
 // while((SYSCTL_PRGPIO_R&0x02) == 0){};// ready?
  GPIO_PORTB_DIR_R &= ~0x0F;       // PB0-3 is an input
```

```
 //GPIO_PORTB_AFSEL_R &= ~0x0F;      // regular port function
 GPIO_PORTB_AMSEL_R &= ~0x0F;      // disable analog on PB0-3
 //GPIO_PORTB_PCTL_R &= ~0x0000FFFF; // PCTL GPIO on PB1
 GPIO_PORTB_DEN_R |= 0x0F;         // PB3-0 enabled as a digital port
        GPIO_PORTB_IS_R &= ~0x0F;                                   // PB 0-3 is edge-
sensitive
        GPIO_PORTB_IBE_R &= ~0x0F;                                  // PB 0-3 is not both edges
        GPIO_PORTB_IEV_R &= ~0x0F;                                  // PB 0-3 falling edge event
        GPIO_PORTB_ICR_R = 0x0F;                                    // clear flag 0-3
        GPIO_PORTB_IM_R |= 0x0F;                                    // arm interrupt on PB
0-3
        //NVIC_PRI0_R = (NVIC_PRI0_R&0xFF00FFFF)|0x00A00000; // (5) priority 5
        NVIC_EN0_R = 0x00000002;                                    //enable interrupt
1(PB) in NVIC
}
// SetTime.c
// Michael Park, Jack Zhao
// Date Created: 02/10/2016
// class for functions to set stopwatch
// Lab Number: 16340
// TA: Mahesh Srinivasan
// Last Revised: 02/15/16

#include <stdio.h>
#include <stdint.h>
#include "SetAlarm.h"
#include "ToggleAlarm.h"
#include "ST7735.h"
#include "Stopwatch.h"
#include "../Shared/tm4c123gh6pm.h"
```

```
extern long StartCritical (void);    // previous I bit, disable interrupts
extern void EndCritical(long sr);    // restore I bit to previous value
extern volatile uint16_t Time_Seconds, Time_Minutes, Time_Hours; //from Main Class
volatile uint16_t StopWatch_Seconds, StopWatch_Minutes, StopWatch_Hours; //from Main Class
volatile uint16_t Stopwatch_Triggered;

void StopWatch_Init(void)
{
        Stopwatch_Triggered=0;
}
void DisplayStopwatch(void)
{

        char AsciiArray[] = {'0','1','2','3','4','5','6','7','8','9'};
        char ch[8];

        ch[0] = AsciiArray[StopWatch_Hours/10];
        ch[1] = AsciiArray[StopWatch_Hours%10];
        ch[2] = ':';
        ch[3] = AsciiArray[StopWatch_Minutes/10];
        ch[4] = AsciiArray[StopWatch_Minutes%10];
        ch[5] = ':';
        ch[6] = AsciiArray[StopWatch_Seconds/10];
        ch[7] = AsciiArray[StopWatch_Seconds%10];

        Output_Clear();
        ST7735_DrawCharS(0, 50, ch[0], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(15, 50, ch[1], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(30, 50, ch[2], ST7735_YELLOW, ST7735_BLACK, 3);
```

```
        ST7735_DrawCharS(45, 50, ch[3], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(60, 50, ch[4], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(75, 50, ch[5], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(90, 50, ch[6], ST7735_YELLOW, ST7735_BLACK, 3);
        ST7735_DrawCharS(105, 50, ch[7], ST7735_YELLOW, ST7735_BLACK, 3);


        //StopWatch_Seconds=0;
        //StopWatch_Minutes=0;
        //StopWatch_Hours=0;



}

void triggerStopWatch(void){
        StopWatch_Seconds=Time_Seconds;
        StopWatch_Minutes=Time_Minutes;
        StopWatch_Hours=Time_Hours;
}
void endStopWatch(void){
        StopWatch_Seconds=Time_Seconds-StopWatch_Seconds;
        StopWatch_Minutes=Time_Minutes-StopWatch_Minutes;
        StopWatch_Hours=Time_Hours-StopWatch_Hours;
}

// Timer1.c
// Runs on LM4F120/TM4C123
// Use TIMER1 in 32-bit periodic mode to request interrupts at a periodic rate
// Daniel Valvano
```

```
// May 5, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
  Program 7.5, example 7.6

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#include "Timer1.h"
#include "TimeDisplay.h"

#define SECONDS_TIME            0x3c
#define MINUTES_TIME            0x3c
#define HOURS_TIME              0x17

void (*PeriodicTask)(void);   // user function

volatile uint16_t Time_Seconds = 0;
```

```
volatile uint16_t Time_Minutes = 0;
volatile uint16_t Time_Hours = 0;
volatile uint8_t displayFlag;

// ****************** 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
}

void Timer1A_Handler(void){
  TIMER1_ICR_R = TIMER_ICR_TATOCINT;           // acknowledge TIMER1A timeout
          Time_Seconds++;
                  //Add 1 Second when the interrupt is triggered.
          displayFlag = 0x01;
```

```
        if(Time_Seconds>=SECONDS_TIME){
                Time_Minutes++;
                Time_Seconds=0;
                displayFlag = 0x03;
        }
        if(Time_Minutes>=MINUTES_TIME){
                Time_Hours++;
                Time_Minutes=0;
                displayFlag = 0x07;
        }
        if(Time_Hours>=HOURS_TIME){
                Time_Hours=0;
        }
}
// Timer1.c
// Runs on LM4F120/TM4C123
// Use TIMER1 in 32-bit periodic mode to request interrupts at a periodic rate
// Daniel Valvano
// May 5, 2015

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015
  Program 7.5, example 7.6

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
```

```
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"
#include "Timer1.h"
#include "TimeDisplay.h"

#define SECONDS_TIME            0x3c
#define MINUTES_TIME            0x3c
#define HOURS_TIME              0x17

void (*PeriodicTask)(void);   // user function

volatile uint16_t Time_Seconds = 0;
volatile uint16_t Time_Minutes = 0;
volatile uint16_t Time_Hours = 0;
volatile uint8_t displayFlag;

// ***************** 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
}

void Timer1A_Handler(void){
  TIMER1_ICR_R = TIMER_ICR_TATOCINT;           // acknowledge TIMER1A timeout
        Time_Seconds++;
                //Add 1 Second when the interrupt is triggered.
        displayFlag = 0x01;

        if(Time_Seconds>=SECONDS_TIME){
                Time_Minutes++;
                Time_Seconds=0;
                displayFlag = 0x03;
        }
        if(Time_Minutes>=MINUTES_TIME){
                Time_Hours++;
                Time_Minutes=0;
                displayFlag = 0x07;
        }
```

```
        if(Time_Hours>=HOURS_TIME){
                Time_Hours=0;
        }
}
// Timer0A.c
// Runs on LM4F120/TM4C123
// Use Timer0A in periodic mode to request interrupts at a particular
// period.
// Daniel Valvano
// September 11, 2013

/* This example accompanies the book
  "Embedded Systems: Introduction to ARM Cortex M Microcontrollers"
  ISBN: 978-1469998749, Jonathan Valvano, copyright (c) 2015
  Volume 1, Program 9.8

 "Embedded Systems: Real Time Interfacing to ARM Cortex M Microcontrollers",
  ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2014
  Volume 2, Program 7.5, example 7.6

 Copyright 2015 by Jonathan W. Valvano, valvano@mail.utexas.edu
   You may use, edit, run or distribute this file
   as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS, IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL, INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
```

```
 */
#include <stdint.h>
#include "../Shared/tm4c123gh6pm.h"



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
//void (*PeriodicTask)(void);   // user function

extern volatile uint16_t toggleSound;

#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);
}

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