Ali Tejani, amt3639

Caroline Yao, chy253

Lab 3 Preparation

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, students 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. There will be switches and a variable resistor. The system will be built on a solderless breadboard and run on the usual USB power. The system may use the on board switches and/or the on board LEDs. Alternatively, the system may include external switches. The speaker will be external. 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?

EE445L students are the engineers and the TA is the client. Students are expected to modify this document to clarify exactly what they plan to build. Students are allowed to divide responsibilities of the project however they wish, but, at the time of demonstration, both students are expected to understand all aspects of the design.

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: estimate of operation time of a battery powered system by dividing energy storage by average current required to run the system

Device driver: A collection of software routines that perform IO functions

Critical section: Locations in a software module, which if an interrupt were to occur at one of these locations, an error would occur

Latency: Response time of a computer to external events

Time jitter: Noise in time measurement

Modular programming: Style of software development that divides the software problem into distinct and independent modules

1.6. Security: How will intellectual property be managed?

The system may include software from Tivaware and from the book. No software written for this project may be transmitted, viewed, or communicated with any other EE445L student past, present, or future (other than the lab partner of course). It is the responsibility of the team to keep its EE445L lab solutions secure.

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 or a keypad. 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 two to 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. The user should be able to set the time (hours, minutes) and be able to set the alarm (hour, minute) using the variable resistor. After some amount of inactivity the system reverts to the main menu. The user should be about to control some aspects of the display configuring the look and feel of the device. The switches MUST 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 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 sound can be a simple square wave. The sound amplitude will be 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 the due date listed in the syllabus. This report includes the final requirements document.

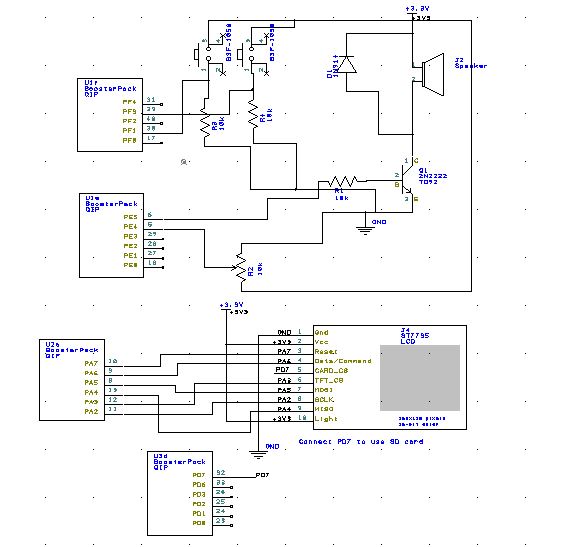
3.2. Audits: How will the clients evaluate progress?

The preparation is due at the beginning of the lab period on the date listed in the syllabus.

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

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

PCB Circuit



// \*\*\*\*\*\*\*\* **ClockMain**.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Main to test modules for clock

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

#include <stdint.h>

#include "ST7735.h"

#include "PLL.h"

#include "SysTickDriver.h"

#include "SpeakerDriver.h"

#include "ST7735.h"

#include "SwitchDriver.h"

#include "ADCSWTrigger.h"

#include "../inc/tm4c123gh6pm.h"

// state values

const uint32\_t DISPLAY\_CLOCK = 0;

const uint32\_t SET\_TIME\_HOUR = 1;

const uint32\_t SET\_TIME\_MINUTE = 2;

const uint32\_t SET\_ALARM\_HOUR = 3;

const uint32\_t SET\_ALARM\_MINUTE = 4;

static uint32\_t State = DISPLAY\_CLOCK; // current state

static uint32\_t AlarmOn = 0; // 1 if alarm is on

static uint32\_t AlarmTime = 0; // time of alarm

static volatile uint32\_t ADCValue = 0; // adc value

extern uint32\_t AtomicTime; // current time

// start set time state, or move to next state

void button1SetTime(void) {

if(State != SET\_TIME\_HOUR && State != SET\_TIME\_MINUTE) {

State = SET\_TIME\_HOUR;

} else {

State = DISPLAY\_CLOCK;

}

}

// start set alarm state, or move to next state

void button2SetAlarm(void) {

if(State != SET\_ALARM\_HOUR && State != SET\_ALARM\_MINUTE) {

State = SET\_ALARM\_HOUR;

} else {

State = DISPLAY\_CLOCK;

}

}

// enable/disable alarm

void buttton3ToggleAlarm(void) {

AlarmOn = 1 - AlarmOn;

}

// move to next state

void button4ChangeMode(void) {

if(State == SET\_TIME\_HOUR) {

State = SET\_TIME\_MINUTE;

} else if(State == SET\_TIME\_MINUTE) {

State = DISPLAY\_CLOCK;

// set AtomicTime

} else if(State == SET\_ALARM\_HOUR) {

State = SET\_ALARM\_MINUTE;

} else if(State == SET\_ALARM\_MINUTE) {

State = DISPLAY\_CLOCK;

// set AlarmTime

}

}

// draw clock and lines

void drawClock(char\* title, uint32\_t time) {

// clear old values

// draw title

ST7735\_SetCursor(0,0);

ST7735\_OutString(title);

ST7735\_OutString("\rAlarm ");

if(AlarmOn) {

ST7735\_OutString("On");

} else {

ST7735\_OutString("Off");

}

// draw analog clock face

// draw digital clock

ST7735\_OutUDec(time/100);

ST7735\_OutString(":");

ST7735\_OutUDec(time%100);

}

int main(void) {

PLL\_Init(Bus80MHz); // init modules

SysTick\_Init();

Speaker\_Init();

ST7735\_InitR(INITR\_REDTAB);

Switch\_Init(button1SetTime, button2SetAlarm, buttton3ToggleAlarm,

button4ChangeMode);

ADC0\_InitSWTriggerSeq3\_Ch9();

while(1) {

uint32\_t time = AtomicTime;

if(State == DISPLAY\_CLOCK) { // draw clock with time from systick

drawClock("Clock", time);

} else if(State == SET\_TIME\_HOUR) { // draw clock with time from start time

// from systick offset by adc

// sample adc

uint32\_t hour = ADCValue / 24;

time = hour \*100 + time%100;

drawClock("Set Time - Hours", time);

} else if(State == SET\_TIME\_MINUTE) { // draw clock with time from start time

// from systick offset by adc

// sample adc

uint32\_t minute = ADCValue / 60;

time = time /100 + minute;

drawClock("Set Time - Minutes",time);

} else if(State == SET\_ALARM\_HOUR) { // draw clock with alarm time offset by adc

uint32\_t hour = ADCValue / 24;

time = hour /100 + AlarmTime%100;

drawClock("Set Alarm - Hours",time);

} else if(State == SET\_ALARM\_MINUTE) { // draw clock with alarm time offset by adc

uint32\_t minute = ADCValue / 60;

time = AlarmTime /100 + minute;

drawClock("Set Alarm - Minutes",time);

}

}

}

// \*\*\*\*\*\*\*\* **SysTickDriver**.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Creates a clock timer which increments every second

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

#ifndef \_\_SYSTICKDRIVER\_H\_\_ // do not include more than once

#define \_\_SYSTICKDRIVER\_H\_\_

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*SysTick\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initialize Systick periodic interrupts

// Input: interrupt period

// Units of period are 12.5ns (assuming 80 MHz clock)

// Maximum is 2^24-1

// Minimum is determined by lenght of ISR

// Output: none

void SysTick\_Init(void);

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*SysTick\_Handler\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Interrupt service routine

// increment seconds, minutes, and hours at appropriate time

// Executed every second

void SysTick\_Handler(void);

#endif // \_\_SYSTICKINTS\_H\_\_

// \*\*\*\*\*\*\*\* **SysTickDriver**.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Creates a clock timer which increments every second

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

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 "../inc/tm4c123gh6pm.h"

#include "SysTickDriver.h"

#define PF2 (\*((volatile uint32\_t \*)0x40025010))

static uint32\_t Seconds;

static uint32\_t Minutes;

static uint32\_t Hours;

volatile uint32\_t AtomicTime;

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

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*SysTick\_Init\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Initialize SysTick periodic interrupts

// Input: interrupt period

// Units of period are 12.5ns (assuming 50 MHz clock)

// Maximum is 2^24-1

// Minimum is determined by length of ISR

// Output: none

void SysTick\_Init(){long sr;

sr = StartCritical();

NVIC\_ST\_CTRL\_R = 0; // disable SysTick during setup

NVIC\_ST\_RELOAD\_R = 79999999;// reload value

NVIC\_ST\_CURRENT\_R = 0; // any write to current clears it

NVIC\_SYS\_PRI3\_R = (NVIC\_SYS\_PRI3\_R&0x00FFFFFF)|0x40000000; // priority 2

// enable SysTick with core clock and interrupts

NVIC\_ST\_CTRL\_R = 0x07;

Seconds = 0;

Minutes = 0;

Hours = 0;

AtomicTime = 0;

EndCritical(sr);

}

// \*\*\*\*\*\*\*\*\*\*\*\*\*\*SysTick\_Handler\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Interrupt service routine

// increment seconds, minutes, and hours at appropriate time

// Executed every second

void SysTick\_Handler(void) {

PF2 ^= 0x04; // heartbeat

Seconds = (Seconds + 1) % 60; // increment seconds

if (Seconds == 0) {

Minutes = (Minutes + 1) % 60; // increment minutes if 60 seconds have passed

if(Minutes == 0) {

Hours = (Hours + 1) % 24; // increment hours if 60 minutes have passed

}

}

AtomicTime = Hours\*100 + Minutes;

}

// \*\*\*\*\*\*\*\* **SpeakerDriver**.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Driver to interface with speaker

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

#ifndef \_\_SPEAKERDRIVER\_H\_\_ // do not include more than once

#define \_\_SPEAKERDRIVER\_H\_\_

// initialize PE5 to output pulse to speaker

// start timer to play A note

void Speaker\_Init(void);

// toggles value at PE5, creating square wave

void Timer2A\_Handler(void);

#endif // \_\_SPEAKERDRIVER\_H\_\_

// \*\*\*\*\*\*\*\* **SpeakerDriver**.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Driver to interface with speaker

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

#include "../inc/tm4c123gh6pm.h"

#include <stdint.h>

#include "SpeakerDriver.h"

#define PE5 (\*((volatile uint32\_t \*)0x40024080))

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

// starts timer 2 and interrupts

// sets timer to play A note

static void Timer2\_Init(void){

volatile uint32\_t delay;

DisableInterrupts();

// \*\*\*\* general initialization \*\*\*\*

SYSCTL\_RCGCTIMER\_R |= 0x04; // activate timer0

delay = SYSCTL\_RCGCTIMER\_R; // allow time to finish activating

TIMER2\_CTL\_R &= ~TIMER\_CTL\_TAEN; // disable timer0A during setup

TIMER2\_CFG\_R = 0; // configure for 32-bit timer mode

// \*\*\*\* timer0A initialization \*\*\*\*

// configure for periodic mode

TIMER2\_TAMR\_R = TIMER\_TAMR\_TAMR\_PERIOD;

TIMER2\_TAILR\_R = 90908; // start value for 880 Hz interrupts

TIMER2\_IMR\_R |= TIMER\_IMR\_TATOIM;// enable timeout (rollover) interrupt

TIMER2\_ICR\_R = TIMER\_ICR\_TATOCINT;// clear timer0A timeout flag

TIMER2\_CTL\_R |= TIMER\_CTL\_TAEN; // enable timer0A 32-b, periodic, interrupts

// \*\*\*\* interrupt initialization \*\*\*\*

// Timer2=priority 1

NVIC\_PRI5\_R = (NVIC\_PRI5\_R&0x00FFFFFF)|0x20000000; // top 3 bits

NVIC\_EN0\_R = 1<<23; // enable interrupt 19 in NVIC

}

// initialize PE5 to output pulse to speaker

// start timer

void Speaker\_Init(void) {

volatile unsigned long delay;

SYSCTL\_RCGCGPIO\_R |= 0x00000010; // enable port E

delay = SYSCTL\_RCGCGPIO\_R;

GPIO\_PORTE\_DIR\_R |= 0x20; // Make PE5 in

GPIO\_PORTE\_AFSEL\_R &= ~0x20; // Disable Alternate Function on PE5

GPIO\_PORTE\_DEN\_R |= 0x20; // Enable digital I/O for PE5

GPIO\_PORTE\_AMSEL\_R &= ~0x20; // Disable analog functionality

Timer2\_Init();

}

// toggles value at PE5, creating square wave

void Timer2A\_Handler(void) {

TIMER2\_ICR\_R = 0x01; // acknowledge timer2a timeout

PE5 ^= 0x20;

}

// \*\*\*\*\*\*\*\* **SwitchDriver**.h \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Driver to interface with speaker

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

// Initialize switch interface on PF4

// Inputs: pointer to a function to call on touch (falling edge),

// pointer to a function to call on release (rising edge)

// Outputs: none

void Switch\_Init(void(\*button1Task)(void), void(\*button2Task)(void),

void(\*button3Task)(void), void(\*button4Task)(void));

// Interrupt on rising or falling edge of PF4 (CCP0)

void GPIOPortF\_Handler(void);

// Interrupt 10 ms after rising edge of PF4

void Timer0A\_Handler(void);

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitPress1(void);

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitRelease1(void);

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitPress2(void);

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitRelease2(void);

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitPress3(void);

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitRelease3(void);

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitPress4(void);

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch\_WaitRelease4(void);

// \*\*\*\*\*\*\*\* **SwitchDriver**.c \*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Ali Tejani and Caroline Yao

// amt3639 and chy253

// Creation Date: 2/8/2017

// Driver to interface with speaker

// Runs on TM4C123

// Lab section: Tue/Thur 12:30 - 2 PM

// TA: Lavanya

// Last Revision: 2/9/2017

/\* This example accompanies the book

"Embedded Systems: Real Time Interfacing to Arm Cortex M Microcontrollers",

ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2015

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/

\*/

// PF4 connected to a negative logic switch using internal pull-up (trigger on both edges)

#include <stdint.h>

#include "SwitchDriver.h"

#include "../inc/tm4c123gh6pm.h"

#define PF0 (\*((volatile uint32\_t \*)0x40025004))

#define PF1 (\*((volatile uint32\_t \*)0x40025008))

#define PF3 (\*((volatile uint32\_t \*)0x40025020))

#define PF4 (\*((volatile uint32\_t \*)0x40025040))

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

volatile static unsigned long Touch1; // true on touch button 1

volatile static unsigned long Release1; // true on release button 1

volatile static unsigned long Touch2; // true on touch button 2

volatile static unsigned long Release2; // true on release button 2

volatile static unsigned long Touch3; // true on touch button 3

volatile static unsigned long Release3; // true on release button 3

volatile static unsigned long Touch4; // true on touch button 4

volatile static unsigned long Release4; // true on release button 4

volatile static unsigned long Last; // previous

void (\*Button1Task)(void); // user function to be executed on button 1 touch

void (\*Button2Task)(void); // user function to be executed on button 2 touch

void (\*Button3Task)(void); // user function to be executed on button 3 touch

void (\*Button4Task)(void); // user function to be executed on button 4 touch

// arm timer 0 to time 10ms in oneshot mode

static void Timer0Arm(void){

TIMER0\_CTL\_R = 0x00000000; // 1) disable TIMER0A during setup

TIMER0\_CFG\_R = 0x00000000; // 2) configure for 32-bit mode

TIMER0\_TAMR\_R = 0x0000001; // 3) 1-SHOT mode

TIMER0\_TAILR\_R = 160000; // 4) 10ms 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

}

// arm gpio interrupts

static void GPIOArm(void){

GPIO\_PORTF\_ICR\_R = 0x1B; // (e) clear flag4

GPIO\_PORTF\_IM\_R |= 0x1B; // (f) arm interrupt on PF4 \*\*\* No IME bit as mentioned in Book \*\*\*

NVIC\_PRI7\_R = (NVIC\_PRI7\_R&0xFF00FFFF)|0x00A00000; // (g) priority 5

NVIC\_EN0\_R = 0x40000000; // (h) enable interrupt 30 in NVIC

}

// Initialize switch interface on PF4,3,1,0

// Inputs: pointer to a function to call on touch (falling edge),

// pointer to a function to call on release (rising edge)

// Outputs: none

void Switch\_Init(void(\*button1Task)(void), void(\*button2Task)(void),

void(\*button3Task)(void), void(\*button4Task)(void)){

// \*\*\*\* general initialization \*\*\*\*

SYSCTL\_RCGCGPIO\_R |= 0x00000020; // (a) activate clock for port F

while((SYSCTL\_PRGPIO\_R & 0x00000020) == 0){};

GPIO\_PORTF\_DIR\_R &= ~0x1B; // (c) make PF4 in (built-in button)

GPIO\_PORTF\_AFSEL\_R &= ~0x1B; // disable alt funct on PF4,3,1,0

GPIO\_PORTF\_DEN\_R |= 0x1B; // enable digital I/O on PF4,3,1,0

GPIO\_PORTF\_PCTL\_R &= ~0x000FF0FF; // configure PF4 as GPIO

GPIO\_PORTF\_AMSEL\_R = 0; // disable analog functionality on PF

GPIO\_PORTF\_PUR\_R |= 0x11; // enable weak pull-up on PF4, pf1

GPIO\_PORTF\_IS\_R &= ~0x1B; // (d) PF4 is edge-sensitive

GPIO\_PORTF\_IBE\_R |= 0x1B; // PF4 is both edges

GPIOArm();

SYSCTL\_RCGCTIMER\_R |= 0x01; // 0) activate TIMER0

Button1Task = button1Task; // user defined methods

Button2Task = button2Task;

Button3Task = button3Task;

Button4Task = button4Task;

Touch1 = 0; // touch and release signals

Release1 = 0;

Touch2 = 0;

Release2 = 0;

Touch3 = 0;

Release3 = 0;

Touch4 = 0;

Release4 = 0;

Last = PF4 | PF3 | PF1 | PF0; // initial switch state

}

// Interrupt on rising or falling edge of PF4 (CCP0)

void GPIOPortF\_Handler(void){

GPIO\_PORTF\_IM\_R &= ~0x10; // disarm interrupt on PF4

if(Last & 0x10) {

Touch4 = 1;

} else {

Release4 = 1;

}

if(Last & 0x8) {

Touch3 = 1;

} else {

Release3 = 1;

}

if(Last & 0x2) {

Touch2 = 1;

} else if(!(Last & 0x2)) {

Release2 = 1;

}

if(Last & 0x1) {

Touch1 = 1;

} else {

Release1 = 1;

}

if(Last == 0x10){ // button 4 press

(\*Button1Task)(); // execute user task

} else if(Last == 0x8){ // button 3 press

(\*Button2Task)(); // execute user task

} else if(Last == 0x2){ // button 2 press

(\*Button3Task)(); // execute user task

} else if(Last == 0x1){ // button 1 press

(\*Button4Task)(); // execute user task

}

Timer0Arm(); // start one shot

}

// Interrupt 10 ms after rising edge of PF4

void Timer0A\_Handler(void){

TIMER0\_IMR\_R = 0x00000000; // disarm timeout interrupt

Last = PF4 | PF3 | PF1 | PF0; // switch state

GPIOArm(); // start GPIO

}

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch1\_WaitPress(void){

while(Touch1==0){}; // wait for press

Touch1 = 0; // set up for next time

}

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch1\_WaitRelease(void){

while(Release1==0){}; // wait

Release1 = 0; // set up for next time

}

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch2\_WaitPress(void){

while(Touch2==0){}; // wait for press

Touch2 = 0; // set up for next time

}

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch2\_WaitRelease(void){

while(Release2==0){}; // wait

Release2 = 0; // set up for next time

}

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch3\_WaitPress(void){

while(Touch3==0){}; // wait for press

Touch3 = 0; // set up for next time

}

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch3\_WaitRelease(void){

while(Release3==0){}; // wait

Release3 = 0; // set up for next time

}

// Wait for switch to be pressed

// There will be minimum time delay from touch to when this function returns

// Inputs: none

// Outputs: none

void Switch4\_WaitPress(void){

while(Touch4==0){}; // wait for press

Touch4 = 0; // set up for next time

}

// Wait for switch to be released

// There will be minimum time delay from release to when this function returns

// Inputs: none

// Outputs: none

void Switch4\_WaitRelease(void){

while(Release4==0){}; // wait

Release4 = 0; // set up for next time

}

// **ST7735**.h excerpt

//\*\*\*\*\*\*\*\*\*\*\*\*\* ST7735\_Line\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Draws one line on the ST7735 color LCD

// Inputs: (x1,y1) is the start point

// (x2,y2) is the end point

// x1,x2 are horizontal positions, columns from the left edge

// must be less than 128

// 0 is on the left, 126 is near the right

// y1,y2 are vertical positions, 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\_Line(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2,

uint16\_t color);

// **ST7735**

//\*\*\*\*\*\*\*\*\*\*\*\*\* ST7735\_Line\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Draws one line on the ST7735 color LCD

// Inputs: (x1,y1) is the start point

// (x2,y2) is the end point

// x1,x2 are horizontal positions, columns from the left edge

// must be less than 128

// 0 is on the left, 126 is near the right

// y1,y2 are vertical positions, 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\_Line(uint16\_t x1, uint16\_t y1, uint16\_t x2, uint16\_t y2, uint16\_t color) {

uint16\_t\* loX = 0; // find point with lower x value

uint16\_t\* loY = 0;

uint16\_t\* hiX = 0;

uint16\_t\* hiY = 0;

if(x1 < x2) {

loX = &x1;

loY = &y1;

hiX = &x2;

hiY = &y2;

} else {

loX = &x2;

loY = &y2;

hiX = &x1;

hiY = &y1;

}

int32\_t slope = ((\*hiY-\*loY)\*128)/(\*hiX-\*loX);

int i;

for(i = \*loX; i <= \*hiX; i += 1) {

int absSlope = slope; // absolute value of scope

int negative = 1; // multiplier for y if \*hiX < \*loX

if(slope < 0) {

absSlope = -1\*slope;

negative = -1;

}

for(int j = 0; j < absSlope; j += 1) { // display

int y = ((slope\*(i - \*loX))/128) + \*loY +j/128\*negative;

ST7735\_DrawPixel(i,y, color);

if(y == \*hiY) { break; }

}

}

}