ECE 314 – Lab 2 Report

Connor Winiarczyk & Hayden Dodge

09/13/2018

**main.c:**

/\*

\* File: touch\_main.c

\* Author: watkinma

\*

\* Description: Prints out the detected position (x, y) of a press on the

\* touchscreen as well as the pressure (z).

\*

\* IMPORTANT: For this example to work, you need to make the following

\* connections from the touchpad to the PIC32:

\* Y+ => AN1 (Pin 3)

\* Y- => RA3 (Pin 10)

\* X+ => RA4 (Pin 12)

\* X- => AN0 (Pin 2)

\*

\*/

#include "config.h"

#include "plib.h"

#include "xc.h"

#include "adc\_intf.h"

#include "TouchScreen.h"

#include "Utilities/Adafruit\_2\_4\_LCD\_Serial\_Library/tft\_master.h"

#include "Utilities/Adafruit\_2\_4\_LCD\_Serial\_Library/tft\_gfx.h"

#include "screen.c"

#define XM AN0

#define YP AN1

typedef struct {

int16\_t color;

int size;

} Brush;

Brush brush = {

.color = 0,

.size = 7

};

// flips input axis

// scale input to screen dimensions

struct TSPoint map\_touch\_to\_pixels(struct TSPoint p){

struct TSPoint np;

np.z = p.z;

np.y = ((p.x - 150) \* (SCREEN\_HEIGHT))/(900 - 150);

np.x = SCREEN\_WIDTH - (((p.y - 350) \* (SCREEN\_WIDTH))/(950 - 350));

return np;

}

// paints a point on the screen based on current brush properties

void paint(TSPoint point) {

// don't paint anything if we are within this rectangle

if(

point.y > SCREEN\_HEIGHT - (30 \* (colorCount + 1)) - (3\*brush.size)

&& point.x > SCREEN\_WIDTH - 30 - (3\*brush.size)

) return;

// do the painting

tft\_fillCircle(point.x, point.y, brush.size, brush.color);

}

int main(int argc, char\*\* argv) {

// legacy setup

SYSTEMConfigPerformance(PBCLK);

configureADC();

// initialize screen

tft\_init\_hw();

tft\_begin();

tft\_setRotation(3);

tft\_fillScreen(ILI9341\_WHITE);

// create buttons for color selection

int i;

for(i = 0; i < colorCount; i++){

Button\* new = newButton(SCREEN\_WIDTH - 30, SCREEN\_HEIGHT - (30 \* (i + 1)), 25, 25);

new -> color = colors[i];

new -> active = (i == 0); // set first button to active

}

// set default color

brush.color = colors[0];

// make a clear screen button

Button\* clear\_screen = newButton(SCREEN\_WIDTH - 30, SCREEN\_HEIGHT - (30 \* (colorCount + 1)), 25, 25);;

clear\_screen -> active = 1; // we want this to also be active

clear\_screen -> color = 0xffff; // we also want it to be white

draw\_buttons();

while(1) {

// get touch point

struct TSPoint p = {0, 0, 0};

getPoint(&p);

// map touch point to screen point

struct TSPoint np;

np = map\_touch\_to\_pixels(p);

// check to see if the user is touching the screen

if(np.z > THRESHOLD) {

//get the button that is being pressed, or NULL if none

Button\* pressed = detectPress(&np);

if(pressed == NULL) paint(np); // no button is pressed, paint

else {

// process button logic

if(pressed == clear\_screen) {

tft\_fillScreen(ILI9341\_WHITE);

} else {

switchTo(pressed);

clear\_screen -> active = 1; // keep clear screen active (with boarder)

brush.color = pressed -> color;

}

draw\_buttons();

}

}

delay\_ms(1);

}

return (EXIT\_SUCCESS);

}

**screen.c:**

#include "TouchScreen.h"

#include <stdint.h>

#include <stdio.h>

#define MAX\_BUTTONS 10

#define THRESHOLD 1

#define SCREEN\_WIDTH 320

#define SCREEN\_HEIGHT 240

#define DEFAULT\_COLOR 0x8bef

typedef struct TSPoint TSPoint;

typedef struct {

int16\_t x, y, width, height, color, active;

} Button;

int colorCount = 6;

int16\_t colors[10] = {0x0000, 0xf800, 0x07e0, 0x001f, 0x07ff, 0xffe0};

static int buttonCount = 0;

static Button buttons[MAX\_BUTTONS];

// makes button and adds it to management array

Button\* newButton(int16\_t x, int16\_t y, int16\_t width, int16\_t height){

if(buttonCount >= MAX\_BUTTONS - 1) return NULL; // NO BUTTON FOR YOU! NEXT!

Button output;

output.x = x;

output.y = y;

output.width = width;

output.height = height;

output.color = DEFAULT\_COLOR;

buttons[buttonCount] = output;

buttonCount ++;

return &buttons[buttonCount - 1];

}

// switches the active state of butons to display only one iwth boarder

void switchTo(Button\* button) {

int i;

for(i=0; i<buttonCount; i++) {

buttons[i].active = 0;

}

button -> active = 1;

}

// returns true if point is within bounds of button

int contains(Button\* button, TSPoint\* point) {

return point -> x > button -> x

&& point -> x < (button -> x + button -> width)

&& point -> y > button -> y

&& point -> y < (button -> y + button -> height);

}

// returns pointer to button that is pressed

// return NULL if none

Button\* detectPress(TSPoint\* input) {

int i;

for(i=0; i< buttonCount; i++) {

if(contains(&(buttons[i]), input)) return &(buttons[i]);

}

return NULL;

}

// draws single button

void draw(Button\* button) {

if(button -> active){

tft\_fillRect(

button -> x - 3,

button -> y - 3,

button -> width + 6,

button -> height + 6,

0x0000

);

}

tft\_fillRect(

button -> x,

button -> y,

button -> width,

button -> height,

button -> color

);

}

// draws all buttons

void draw\_buttons(){

tft\_fillRect(

SCREEN\_WIDTH -33,

SCREEN\_HEIGHT - (30 \* (colorCount + 1)),

33,

30 \* (colorCount + 1),

0xffff

);

int i;

for(i = 0; i < buttonCount; i++){

draw(&buttons[i]);

}

}

We had trouble wiring the screen. We had trouble comprehending the orientation of the x and y axis of the screen. We had numerous problems with the IDE. One of which was importing the Adafruit and touch screen libraries. We were able to compile the project from the command but unable to flash/program the device from the command line. We unfortunately have to keep using the IDE. We may have broken the microstick ii. When we were done with the project, suddenly Unicode letters displayed randomly on the screen. We were unable to remove these Unicode characters until we switched to the spare microstick ii. However, this caused further problems. This broke the touch screen coordinate to display coordinate function. This caused us to recalibrate the screen manually and rewrite the function multiple times. During refactor and code clean, we messed up a few times. The code doesn’t feel as clean as it should. Furthermore, the are a few design changes needed to make the generic button usage more function and less “hacky”