#include <Elegoo\_GFX.h>

#include <MCUFRIEND\_kbv.h>

#include <TouchScreen.h>

#include "HX711.h"

#include <EEPROM.h>

HX711 loadcell;

bool amitesting = false;

bool amitesting2 = false;

bool prevStateTesting = true;

const int LOADCELL\_DOUT\_PIN = 35;

const int LOADCELL\_SCK\_PIN = 27;

const long LOADCELL\_OFFSET = 192500;

const long LOADCELL\_DIVIDER = 181;

bool testStarted = false;

struct EEPROM\_Data{

 float scalingFactor;

 };

EEPROM\_Data eepromData; //Variable to store custom object read from EEPROM.

#if defined(\_\_SAM3X8E\_\_)

 #undef \_\_FlashStringHelper::F(string\_literal)

 #define F(string\_literal) string\_literal

#endif

#define YP A3 // must be an analog pin, use "An" notation!

#define XM A2 // must be an analog pin, use "An" notation!

#define YM 9 // can be a digital pin

#define XP 8 // can be a digital pin

//Touch For New ILI9341 TP

#define TS\_MINX 118

#define TS\_MAXX 916

#define TS\_MINY 86

#define TS\_MAXY 916

// For better pressure precision, we need to know the resistance

// between X+ and X- Use any multimeter to read it

// For the one we're using, its 300 ohms across the X plate

TouchScreen ts = TouchScreen(XP, YP, XM, YM, 300);

#define LCD\_CS A3

#define LCD\_CD A2

#define LCD\_WR A1

#define LCD\_RD A0

// optional

#define LCD\_RESET A4

// Assign human-readable names to some common 16-bit color values:

#define BLACK 0x0000

#define BLUE 0x001F

#define RED 0xF800

#define GREEN 0x07E0

#define CYAN 0x07FF

#define MAGENTA 0xF81F

#define YELLOW 0xFFE0

#define WHITE 0xFFFF

#define GREY 0x9CF3

#define pi 3.14159

#define MINPRESSURE 10

#define MAXPRESSURE 1000

bool newTest = false;

float prevSuction = 0;

const int BORDER\_OFFSET = 10;

int pwm44pin=44;

MCUFRIEND\_kbv tft(LCD\_CS, LCD\_CD, LCD\_WR, LCD\_RD, LCD\_RESET);

#define BOXSIZE 40

#define PENRADIUS 3

int oldColor, currentColor;

unsigned long start\_time;

void start\_timer()

{

 start\_time = millis();

}

bool should\_read\_weight(int timeout){

 if(millis()-start\_time < timeout) return false;

 else {

 start\_time = millis();

 return true;

 }

}

typedef struct tagBUTTON

{

 int x;

 int y;

 int w;

 int h;

 //int radius;

 int color;

 char label[15];

 //int (\*handler)(int index, TSPoint p, int val);

 //int val;

} BUTTON;

unsigned long elapsedTime = 0;

void init\_weight\_scale(void)

{

 // 3. Initialize library

 loadcell.begin(LOADCELL\_DOUT\_PIN, LOADCELL\_SCK\_PIN);

 loadcell.set\_scale(LOADCELL\_DIVIDER);

 loadcell.set\_offset(LOADCELL\_OFFSET);

 loadcell.power\_up();

 loadcell.tare();

}

float weight\_g(){

 return loadcell.get\_units(1)\*eepromData.scalingFactor;

}

void drawContainer(){

 tft.drawRect(BORDER\_OFFSET,BORDER\_OFFSET,tft.width()-2\*BORDER\_OFFSET, tft.height()\*3/5, BLACK);

 tft.fillRect(BORDER\_OFFSET+1, BORDER\_OFFSET, tft.width()-2\*(BORDER\_OFFSET+1), (tft.height()\*3/5-1), CYAN);

 //tft.fillRect(BORDER\_OFFSET

}

//int colorBarArea[100][35];

void drawContainerWithFluid(float\* weightSuctioned){

 tft.drawRect(BORDER\_OFFSET,BORDER\_OFFSET,tft.width()-2\*BORDER\_OFFSET, tft.height()\*3/5, BLACK);

 tft.fillRect(BORDER\_OFFSET+1, BORDER\_OFFSET+1, tft.width()-2\*(BORDER\_OFFSET+1), (tft.height()\*3/5-1), WHITE);

 Serial.println(\*weightSuctioned);

 \*weightSuctioned = (\*weightSuctioned < 1000) ? \*weightSuctioned : 1000;

 Serial.println(\*weightSuctioned);

 \*weightSuctioned = (\*weightSuctioned > 0) ? \*weightSuctioned : 0;

 Serial.println(\*weightSuctioned);

 tft.fillRect(BORDER\_OFFSET+1, BORDER\_OFFSET+(\*weightSuctioned/1000)\*(tft.height()\*3/5-1), tft.width()-2\*(BORDER\_OFFSET+1), (tft.height()\*3/5-1)\*(1000-\*weightSuctioned)/1000, CYAN);

}

void print\_time(float timeToSuck){

 static long lastTime=5555;

 char msg[200];

 char buf[20];

 dtostrf(timeToSuck, 4, 1, buf); //4 is mininum width, 1 value after decimal point

 snprintf(msg,sizeof(msg)," %s ms", buf);

 tft.setCursor(BORDER\_OFFSET+10, tft.height()\*3/5-20);

 tft.setRotation(0);

 tft.setTextColor(YELLOW); tft.setTextSize(3);

 tft.println(msg);

 lastTime = timeToSuck;

}

void print\_weight(float weight){

 static long last\_weight=5555;

 char msg[200];

 char buf[20];

 dtostrf(weight, 4, 1, buf); //4 is mininum width, 1 value after decimal point

 snprintf(msg,sizeof(msg)," %sg", buf);

 if (!amitesting && !amitesting2)

 {

 tft.fillRect(BORDER\_OFFSET+4, tft.height()\*3/5-60, tft.width()-2\*(BORDER\_OFFSET+4), 35, GREY);

 }

 tft.setCursor(BORDER\_OFFSET+10, tft.height()\*3/5-60);

 tft.setRotation(0);

 tft.setTextColor(YELLOW); tft.setTextSize(3);

 tft.println(msg);

 last\_weight = weight;

}

void testNow(){

 drawContainer();

 float amountSuctioned = 0;

 tft.setCursor(BORDER\_OFFSET+10, tft.height()\*3/5-20);

 tft.setRotation(0);

 tft.setTextColor(YELLOW); tft.setTextSize(2);

 delay(1000);

 tft.println("Ready For Suction");

 bool testDone = false;

 if (amitesting)

 {

 start\_timer();

 float initWeight = weight\_g();

 while (!testDone){

 float weightBeginSuck = weight\_g();

 if (weightBeginSuck-initWeight < -2){

 delay(30000);

 amountSuctioned = weightBeginSuck-weight\_g();

 drawContainerWithFluid(&amountSuctioned);

 print\_weight(amountSuctioned);

 testDone = true;

 Serial.println(weightBeginSuck);

 Serial.println(initWeight);

 Serial.println(amountSuctioned);

 }

 else if(millis()-start\_time>20000)

 {

 drawContainer();

 testDone = true;

 tft.setCursor(tft.width()/4, tft.height()\*2/5);

 tft.setRotation(0);

 tft.setTextColor(BLACK); tft.setTextSize(2);

 tft.print("No Suction \n Detected!");

 }

 }

 }

 else

 {

 float initWeight = weight\_g();

 start\_timer();

 unsigned long timeToSuck = 0;

 while(!testDone)

 {

 if ((millis()- start\_time) > 120000)

 {

 testDone = true;

 drawContainer();

 tft.setCursor(tft.width()/4, tft.height()\*2/5);

 tft.setRotation(0);

 tft.setTextColor(BLACK); tft.setTextSize(2);

 tft.print("Test took \n 2 min!");

 break;

 }

 float weightBeginSuck = weight\_g();

 elapsedTime = millis()-start\_time;

 if (weightBeginSuck - initWeight < -2){

 unsigned long beginTime = micros();

 while (!testDone){

 timeToSuck = micros() - beginTime;

 amountSuctioned = weightBeginSuck-weight\_g();

 if (amountSuctioned > 150){

 testDone = true;

 }

 if (timeToSuck > 120000000)

 {

 testDone = true;

 drawContainer();

 tft.setCursor(tft.width()/4, tft.height()\*2/5);

 tft.setRotation(0);

 tft.setTextColor(BLACK); tft.setTextSize(2);

 tft.print("Test took \n 2 min!");

 }

 }

 drawContainerWithFluid(&amountSuctioned);

 print\_time(timeToSuck/1000.0);

 elapsedTime = 0;

 }

 else if (elapsedTime > 120000){

 drawContainer();

 testDone = true;

 drawContainerWithFluid(&amountSuctioned);

 elapsedTime = 0;

 tft.setCursor(tft.width()/4, tft.height()\*2/5);

 tft.setRotation(0);

 tft.setTextColor(BLACK); tft.setTextSize(2);

 tft.print("No Suction \n Detected!");

 }

 }

 }

}

void calibration(){

 eepromData.scalingFactor = 1.0;

 eepromData.scalingFactor = 200.0/weight\_g();

 EEPROM.put(0, eepromData.scalingFactor);

}

void tare\_button(){

 loadcell.tare();

}

void testingSuction(){

 amitesting = true;

 delay(1000);

}

void g\_TestButton(){

 amitesting2 = true;

 delay(1000);

}

void backToHome(){

 amitesting = false;

 amitesting2 = false;

 delay(500);

}

BUTTON buttons[]=

 {

 {BORDER\_OFFSET,225,tft.width()-2\*BORDER\_OFFSET,30,GREEN,"30 SEC TEST"},//&testingSuction, 255},

 {BORDER\_OFFSET,265,75,30,BLUE,"200G"},//,&calibration,0},

 // {0,100,50,30,BLUE,"LOW",&gen\_button,10},

 // {60,100,60,30,RED,"MED",&gen\_button,30},

 // {130,100,60,30,GREEN,"HIGH",&gen\_button,100},

 {130,265,65,30,YELLOW,"TARE"},//&tare\_button,0},

 {BORDER\_OFFSET, 185, tft.width()-2\*BORDER\_OFFSET,30, BLACK,"150 GRAM TEST"},

 {BORDER\_OFFSET, 225, tft.width()-2\*BORDER\_OFFSET,30,GREEN,"TEST NOW"},//&testNow, 255},

 {BORDER\_OFFSET, 265, tft.width()-2\*BORDER\_OFFSET,30,MAGENTA,"BACK TO HOME"}//, &backToHome, 0},

 };

#define NUM\_BUTTONS sizeof(buttons)/sizeof(BUTTON)

BUTTON startTesting = buttons[0];

BUTTON calibrateButton = buttons[1];

BUTTON tareButton = buttons[2];

BUTTON gTestButton = buttons[3];

BUTTON testNowButton = buttons[4];

BUTTON homeButton = buttons[5];

void printTitle(){

 tft.setCursor(tft.width()/5-12, 30);

 tft.setRotation(0);

 tft.setTextColor(RED);

 tft.setTextSize(2);

 tft.println("Suction Tester");

}

void draw\_button(int x, int y, int w, int h, int color, char \*string)

{

 tft.fillRect(x, y, w, h, color);

 tft.drawRect(x, y, w, h, WHITE);

 tft.setCursor(x+10, y+10);

 tft.setRotation(0);

 if(color!=BLACK) tft.setTextColor(BLACK);

 else tft.setTextColor(WHITE);

 tft.setTextSize(2);

 tft.println(string);

}

int check\_buttons(TSPoint p)

{

 int hit=-1;

 if (!amitesting && !amitesting2)

 {

 for(int b=0;b<4;b++)

 {

 BUTTON \*bb=&buttons[b];

 if ( (p.x > bb->x) && (p.x < (bb->x + bb->w))

 && (p.y > bb->y) && (p.y < (bb->y + bb->h)))

 {

 hit=b;

 }

 }

 }

 else

 {

 for(int b=4;b<NUM\_BUTTONS;b++)

 {

 BUTTON \*bb=&buttons[b];

 if ( (p.x > bb->x) && (p.x < (bb->x + bb->w))

 && (p.y > bb->y) && (p.y < (bb->y + bb->h)))

 {

 hit=b;

 }

 }

 }

 return(hit);

}

void setup() {

 // put your setup code here, to run once:

 Serial.begin(9600);

 //Serial.println("Hooray");

 EEPROM.get(0, eepromData.scalingFactor);

 tft.reset();

 init\_weight\_scale();

 //Serial.println(loadcell.get\_units(10));

 //Serial.println("Past initialization of scales");

 uint16\_t identifier = tft.readID();

 //Serial.println("Identifier: ");

 //Serial.println(identifier);

 if(identifier == 0x9325) {

 //Serial.println(F("Found ILI9325 LCD driver"));

 } else if(identifier == 0x9328) {

 //Serial.println(F("Found ILI9328 LCD driver"));

 } else if(identifier == 0x4535) {

 //Serial.println(F("Found LGDP4535 LCD driver"));

 }else if(identifier == 0x7575) {

 //Serial.println(F("Found HX8347G LCD driver"));

 } else if(identifier == 0x9341) {

 //Serial.println(F("Found ILI9341 LCD driver"));

 } else if(identifier == 0x8357) {

 //Serial.println(F("Found HX8357D LCD driver"));

 } else if(identifier==0x0101)

 {

 identifier=0x9341;

 Serial.println(F("Found 0x9341 LCD driver"));

 }else {

 Serial.print(F("Unknown LCD driver chip: "));

 Serial.println(identifier, HEX);

 Serial.println(F("If using the Elegoo 2.8\" TFT Arduino shield, the line:"));

 Serial.println(F(" #define USE\_Elegoo\_SHIELD\_PINOUT"));

 Serial.println(F("should appear in the library header (Elegoo\_TFT.h)."));

 Serial.println(F("If using the breakout board, it should NOT be #defined!"));

 Serial.println(F("Also if using the breakout, double-check that all wiring"));

 Serial.println(F("matches the tutorial."));

 identifier=0x9341;

 }

 Serial.println("Past if-else");

 tft.begin(identifier);

 Serial.println("Past begin");

 tft.setRotation(0);

 tft.fillScreen(BLACK);

 pinMode(13, OUTPUT);

 pinMode(pwm44pin,OUTPUT);

 start\_timer();

 Serial.println("Past startTimer");

}

void loop() {

 digitalWrite(13, HIGH);

 TSPoint p = ts.getPoint();

 TSPoint orig = p;

 digitalWrite(13,LOW);

 pinMode(XM, OUTPUT);

 pinMode(YP, OUTPUT);

 if (!amitesting && !amitesting2){

 if (prevStateTesting)

 {

 delay(500);

 //Serial.println("Hmmmmm");

 //tft.fillScreen(BLACK);

 tft.fillScreen(0x9CF3);// fills it grey

 printTitle();

 draw\_button(startTesting.x, startTesting.y, startTesting.w, startTesting.h, startTesting.color, startTesting.label);

 draw\_button(calibrateButton.x, calibrateButton.y, calibrateButton.w, calibrateButton.h, calibrateButton.color, calibrateButton.label);

 draw\_button(tareButton.x, tareButton.y, tareButton.w, tareButton.h, tareButton.color, tareButton.label);

 draw\_button(gTestButton.x, gTestButton.y, gTestButton.w, gTestButton.h, gTestButton.color, gTestButton.label);

 }

 if (should\_read\_weight(200))

 {

 print\_weight(weight\_g());

 }

 prevStateTesting = false;

 }

 else if (amitesting)

 {

 if (!prevStateTesting)

 {

 delay(500);

 tft.fillScreen(GREY);// fills it grey

 draw\_button(testNowButton.x, testNowButton.y, testNowButton.w, testNowButton.h, testNowButton.color, testNowButton.label);

 draw\_button(homeButton.x, homeButton.y, homeButton.w, homeButton.h, homeButton.color, homeButton.label);

 }

 prevStateTesting = true;

 }

 else

 {

 if (!prevStateTesting)

 {

 delay(500);

 tft.fillScreen(GREY);// fills it grey

 draw\_button(testNowButton.x, testNowButton.y, testNowButton.w, testNowButton.h, testNowButton.color, testNowButton.label);

 draw\_button(homeButton.x, homeButton.y, homeButton.w, homeButton.h, homeButton.color, homeButton.label);

 }

 prevStateTesting = true;

 }

 if (p.z > MINPRESSURE && p.z < MAXPRESSURE)

 {

 p.x = map(p.x, TS\_MINX, TS\_MAXX, tft.width(), 0);

 p.y = tft.height() - map(p.y, TS\_MINY, TS\_MAXY, tft.height(), 0);

 int hit = check\_buttons(p);

 if (hit == 0)

 {

 testingSuction();

 }

 else if (hit == 1)

 {

 calibration();

 }

 else if (hit == 2)

 {

 tare\_button();

 }

 else if (hit == 3)

 {

 g\_TestButton();

 }

 else if (hit == 4)

 {

 testNow();

 }

 else if (hit == 5)

 {

 backToHome();

 }

 }

}