# User Interface Block Validation

Block Champion: Savannah Tanner

Date: Jan-24-2024

## 1. Description

The user interface serves as a bridge between the user and the technology embedded in the fish tank monitor. The user interface allows the user to make choices with the system in order to monitor the health of their aquarium. The block intakes data from the user, sends this data through the user interface code (Figure 3), and returns an output through the screen. The user interface itself operates off of five pages visible to the user including the home, settings, parameter history, inhabitants list, and notification pages. Each of these pages serves a different but interconnected purpose.

The home page provides essential information to the user including the date, time, and temperature of the tank. Additionally, it offers two options for user interaction, enabling the user to indicate whether the fish tank has been cleaned or the fish have been fed. It also includes a notification symbol to indicate that there are new notifications to be checked by the user. This page serves as a quick and crucial start for the user to access information on their tank.

The settings page includes the option for the user to alter the date, time, parameter ranges, and integrated routines. The monitor's routines allow the user to receive notifications to clean their tank or feed their fish, and due to the wide variety of requirements across species, these conditions must be updateable. The automatic testing routine also must be controlled depending on the frequency of testing desired by the user. As the parameter testing is done automatically, this routine controls the machine's testing unit. The settings page also offers the user the opportunity to export their testing data onto an external device.

The parameter history page is designed for users to comprehensively review their test results. The data is presented in both graph and list forms for each parameter, providing a long-term overview of the tank's conditions. This feature enhances the user's ability to track changes in their fish tank over time, contributing to proper fish care and informed decision-making regarding the tank environment.

The inhabitants list page offers a centralized location for users to keep track of the various creatures living in their fish tank. This list contributes to the user's awareness of the tank's population, facilitating better care and management.

The notifications page serves as a critical communication hub to display important notices for the user. It displays alerts for potentially dangerous test results, errors within the system, and information on when the fish tank monitor is running low on test strips. By providing this information in a timely manner, users can stay informed about potential issues within their tank.

The user interface is designed for a rectangular, [e-ink display](https://drive.google.com/file/d/1LCnIXtZZ29pK2zgxB5DYRpUhkZnG7cQx/view?usp=drive_link) which only updates upon request [1]. Because of this, the layout is simple and easy to understand to increase accessibility. Because of the e-ink screen, the display does not use any color and instead displays black text and images on a white background [1].

The block utilizes sensor control code which requires the use of a [color sensor](https://drive.google.com/file/d/11KFGTJVkesoG8jHHQOEfUxUZ5dy5fkVU/view?usp=drive_link) [3] and a [digital thermometer](https://drive.google.com/file/d/1Wr788dnIJMPOlnrYm890tkVMwFFs2bmD/view?usp=drive_link) [2]. These connect to the screen to receive power and provide data which enables the user interface to display water parameters to the user.

The fish tank monitor's user interface is designed to improve and enhance the user experience and facilitate effective management of the fish tank environment. Its user-friendly pages, customization options, data representation, and notification system combine to ensure the well-being of the aquatic ecosystem of the fish tank.

## 2. Design

In the design portion for the user interface block, I have included a black box diagram, a flowchart displaying the flow of the pages, and a wiring description. The black box diagram details the connections between the user and the user interface. This connection entails selections from the user, moving between pages, and commands that the user might use within the system. The next connection is between the user interface and the user output. This represents the screen output to the user allowing them to operate the system. The final connection is between the user interface and the screen data. This connection represents the user interface's control over the output of the screen. The flowchart represents how the user can move about the interface. It shows all five pages, buttons on each page, sub-pages such as the graph and list views of the test results, as well as certain constants displayed on each screen. The code represents the final functions required to operate the user interface. The wiring diagram shows how the color sensor and temperature sensor connect to the screen, though this does not account for the printed circuit board designed by our team.

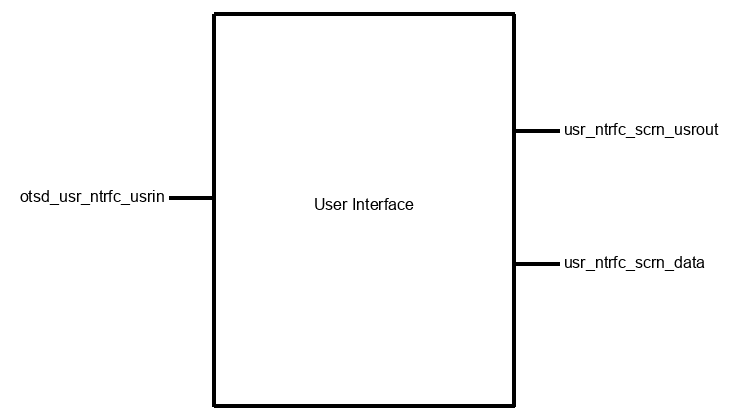
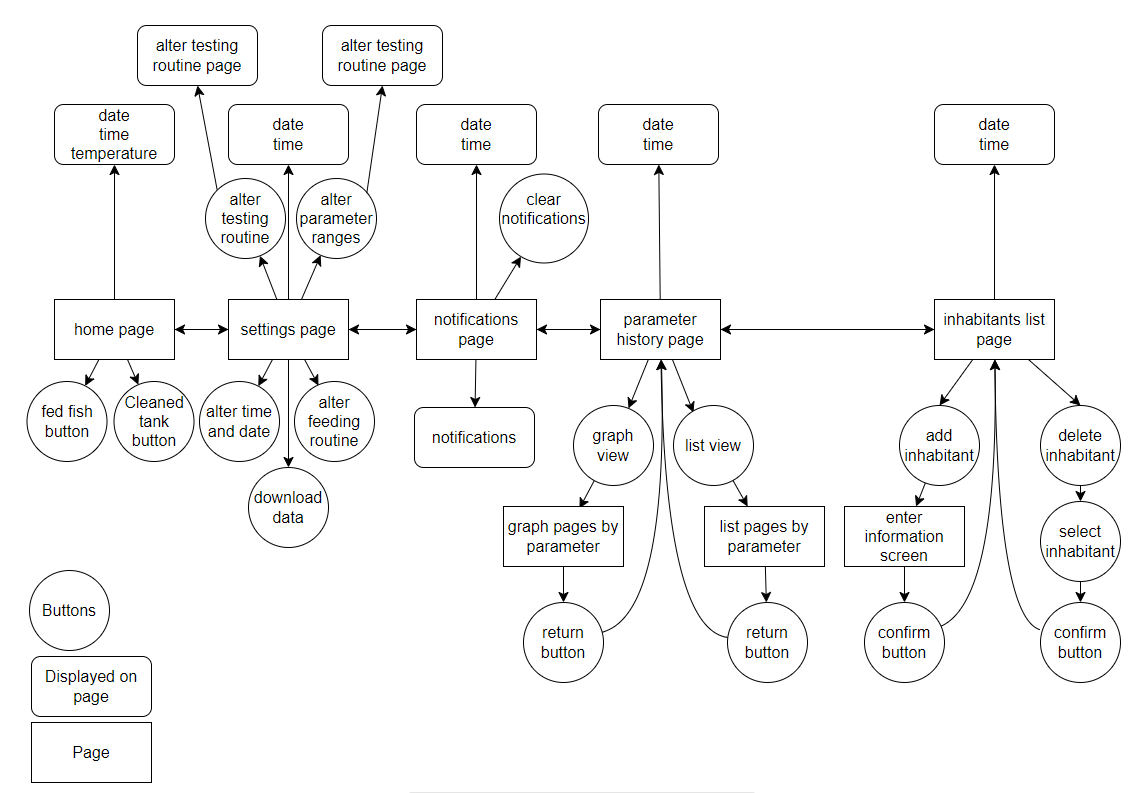


Figure 1: User Interface Block Diagram; Inputs: otsd\_usr\_ntrfc\_usrin; Outputs: usr\_ntrfc\_scrn\_usrout, usr\_ntrfc\_scrn\_data

Figure 2: User Interface State Diagram

// FINAL BLOCK 1 VALIDATION

// USER INTERFACE BLOCK

// AUTHOR: SAVANNAH TANNER

// DATE: 3/10/2024

#include <Wire.h>

#include "Adafruit\_TCS34725.h"

#include <OneWire.h>

#include <DallasTemperature.h>

#include <GxEPD.h>

#include <GxFont\_GFX.h>

#include <Fonts/FreeMonoBold9pt7b.h> // Change the font size here

#include <GxIO/GxIO\_SPI/GxIO\_SPI.h>

#include <GxIO/GxIO.h>

#include <GxDEPG0213BN/GxDEPG0213BN.h>

// Define SPI pin configurations

#define SPI\_MOSI 23

#define SPI\_MISO -1

#define SPI\_CLK 18

// Define E-ink display pin configurations

#define ELINK\_SS 5

#define ELINK\_BUSY 4

#define ELINK\_RESET 16

#define ELINK\_DC 17

#define TEMPERATURE\_BUS 19 // Digital pin where the DS18B20 is connected

#define SDA\_PIN 21

#define SCL\_PIN 22

#define BUTTON\_PIN 15

#define BUTTON\_THRESHOLD 3

#define POTENTIOMETER\_PIN 13

// Color Sensor Variables

Adafruit\_TCS34725 tcs = Adafruit\_TCS34725(TCS34725\_INTEGRATIONTIME\_614MS, TCS34725\_GAIN\_1X);

// Thermometer Variables

OneWire Temperature(TEMPERATURE\_BUS);

DallasTemperature sensors(&Temperature);

bool tempUnits;

// Initialize SPI and E-ink display objects

GxIO\_Class io(SPI, /\*CS=5\*/ ELINK\_SS, /\*DC=\*/ ELINK\_DC, /\*RST=\*/ ELINK\_RESET);

GxEPD\_Class display(io, /\*RST=\*/ ELINK\_RESET, /\*BUSY=\*/ ELINK\_BUSY);

// Button and Potentiometer variables

int lastButtonState = HIGH; // the previous state from the input pin

int currentButtonState; // the current reading from the input pin

int lastPotValue = 0; // the previous value from the potentiometer

int currentPotValue; // the current reading from the potentiometer

volatile bool buttonPressed = false;

int currentPage = 0;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// PARAMETER COMPARISON FUNCTIONS

// Function to compare RGB values to predefined PH colors

void compareColorToPH(uint16\_t r, uint16\_t g, uint16\_t b) {

// Predefined pH color values

uint16\_t phColors[][3] = {

{236, 148, 0}, // pH 6

{255, 168, 0}, // pH 6.5

{255, 150, 29}, // pH 7

{253, 105, 20}, // pH 7.5

{222, 48, 4}, // pH 8

{226, 26, 11}, // pH 8.5

{230, 4, 28} // pH 9

};

// Calculate the Euclidean distance for each pH color

float minDistance = MAXFLOAT;

int closestPHIndex = -1;

for (int i = 0; i < sizeof(phColors) / sizeof(phColors[0]); i++) {

float distance = sqrt(pow(r - phColors[i][0], 2) + pow(g - phColors[i][1], 2) + pow(b - phColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestPHIndex = i;

}

}

// Display the closest pH level

float closestPH = 6.0 + closestPHIndex \* 0.5;

Serial.print("Detected pH level: ");

Serial.println(closestPH);

}

// Function to compare RGB values to predefined GH colors

void compareColorToGH(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t ghColors[][3] = {

{171, 187, 168}, // GH 0

{118, 165, 162}, // GH 30

{122, 146, 158}, // GH 60

{101, 111, 136}, // GH 120

{147, 130, 177} // GH 180

};

float minDistance = MAXFLOAT;

int closestGHIndex = -1;

for (int i = 0; i < sizeof(ghColors) / sizeof(ghColors[0]); i++) {

float distance = sqrt(pow(r - ghColors[i][0], 2) + pow(g - ghColors[i][1], 2) + pow(b - ghColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestGHIndex = i;

}

}

float closestGH = closestGHIndex \* 30.0;

Serial.print("Detected GH value: ");

Serial.println(closestGH);

}

// Function to compare RGB values to predefined KH colors

void compareColorToKH(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t khColors[][3] = {

{239, 180, 66}, // KH 0

{194, 156, 60}, // KH 40

{181, 173, 112}, // KH 80

{142, 148, 102}, // KH 120

{158, 172, 139} // KH 180

};

float minDistance = MAXFLOAT;

int closestKHIndex = -1;

for (int i = 0; i < sizeof(khColors) / sizeof(khColors[0]); i++) {

float distance = sqrt(pow(r - khColors[i][0], 2) + pow(g - khColors[i][1], 2) + pow(b - khColors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestKHIndex = i;

}

}

float closestKH = closestKHIndex \* 40.0;

Serial.print("Detected KH value: ");

Serial.println(closestKH);

}

// Function to compare RGB values to predefined NO2 colors

void compareColorToNO2(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t no2Colors[][3] = {

{239, 177, 142}, // NO2 0

{250, 182, 150}, // NO2 0.5

{232, 158, 133}, // NO2 1

{227, 127, 116}, // NO2 3

{250, 126, 114} // NO2 5

};

float minDistance = MAXFLOAT;

int closestNO2Index = -1;

for (int i = 0; i < sizeof(no2Colors) / sizeof(no2Colors[0]); i++) {

float distance = sqrt(pow(r - no2Colors[i][0], 2) + pow(g - no2Colors[i][1], 2) + pow(b - no2Colors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestNO2Index = i;

}

}

float closestNO2 = closestNO2Index \* 0.5;

Serial.print("Detected NO2 value: ");

Serial.println(closestNO2);

}

// Function to compare RGB values to predefined NO3 colors

void compareColorToNO3(uint16\_t r, uint16\_t g, uint16\_t b) {

uint16\_t no3Colors[][3] = {

{244, 184, 149}, // NO3 0

{254, 184, 153}, // NO3 20

{248, 165, 146}, // NO3 40

{230, 123, 103}, // NO3 80

{230, 93, 87} // NO3 160

};

float minDistance = MAXFLOAT;

int closestNO3Index = -1;

for (int i = 0; i < sizeof(no3Colors) / sizeof(no3Colors[0]); i++) {

float distance = sqrt(pow(r - no3Colors[i][0], 2) + pow(g - no3Colors[i][1], 2) + pow(b - no3Colors[i][2], 2));

if (distance < minDistance) {

minDistance = distance;

closestNO3Index = i;

}

}

float closestNO3 = closestNO3Index \* 20.0;

Serial.print("Detected NO3 value: ");

Serial.println(closestNO3);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// TEMPERATURE FUNCTIONS

void temperatureSensor() {

sensors.requestTemperatures();

float temperatureC = sensors.getTempCByIndex(0);

if (temperatureC != DEVICE\_DISCONNECTED\_C) {

if (tempUnits == 0){

// Celsius

Serial.print("Temperature: ");

Serial.print(temperatureC);

Serial.println("�C");

}

if (tempUnits == 1){

// Fahrenheit

Serial.print("Temperature: ");

Serial.print((temperatureC \* 9/5) + 32);

Serial.println("�F");

}

Serial.println();

}

else {

Serial.println("Error reading temperature!");

}

}

void chooseFahrenheitCelsius(){

Serial.println("Fahrenheit (F) or Celsius (C): ");

while (!Serial.available()) {

// Wait for user input

}

char choice = Serial.read();

if (choice == 'F'){

tempUnits = 1;

}

else if (choice == 'C'){

tempUnits = 0;

}

else{

Serial.println("Invalid choice. Please enter F for Fahrenheit or C for Celsius.");

chooseFahrenheitCelsius(); // Ask again if the choice is invalid

}

}

0

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// COLOR FUNCTIONS

void printHexColor(uint16\_t r, uint16\_t g, uint16\_t b) {

unsigned long hexColor = ((unsigned long)r << 16) | ((unsigned long)g << 8) | b;

Serial.print("HEX Color: 0x");

Serial.println(hexColor, HEX);

}

void html\_rgb(uint16\_t r, uint16\_t g, uint16\_t b, uint16\_t c, float (&rgb)[3]) {

// Calculate normalization factor

float maxRGB = max(max(r, g), b);

float factor = 255.0 / maxRGB;

// Normalize RGB values

rgb[0] = r \* factor;

rgb[1] = g \* factor;

rgb[2] = b \* factor;

Serial.print("R: "); Serial.print(rgb[0]);

Serial.print(", G: "); Serial.print(rgb[1]);

Serial.print(", B: "); Serial.print(rgb[2]);

Serial.println();

}

String html\_hex(uint16\_t r, uint16\_t g, uint16\_t b, uint16\_t c) {

float rgb[3];

html\_rgb(r, g, b, c, rgb);

return String(String((int)rgb[0], HEX) + String((int)rgb[1], HEX) + String((int)rgb[2], HEX));

}

void colorSensor() {

uint16\_t r, g, b, c, colorTemp, lux;

tcs.getRawData(&r, &g, &b, &c);

colorTemp = tcs.calculateColorTemperature(r, g, b);

lux = tcs.calculateLux(r, g, b);

Serial.print("R: "); Serial.print(r);

Serial.print(", G: "); Serial.print(g);

Serial.print(", B: "); Serial.print(b);

Serial.println();

String hexColor = html\_hex(r, g, b, c);

float rgb[3];

html\_rgb(r, g, b, c, rgb);

Serial.print("HEX Color: 0x"); Serial.println(hexColor);

compareColorToPH(r, g, b); // call the PH function with corrected values

compareColorToGH(r, g, b); // Call the GH function with corrected values

compareColorToKH(r, g, b); // Call the KH function with corrected values

compareColorToNO2(r, g, b); // Call the NO2 function with corrected values

compareColorToNO3(r, g, b); // Call the NO3 function with corrected values

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// INTERRUPT FOR BUTTON PRESS

void buttonISR() {

buttonPressed = true;

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// DISPLAY FUNCS

bool leftRightFunc(int currentPotValue){

bool leftRight;

int threshold = 4095 / 2;

if (currentPotValue > threshold) {

leftRight = true; // 1 represents true

} else {

leftRight = false; // 0 represents false

}

return leftRight;

}

void switchPages(bool leftRight){

if (leftRight) {

currentPage += 1;

if (currentPage >= 4){

currentPage = 0;

}

} else {

currentPage -= 1;

if (currentPage <= -1){

currentPage = 3;

}

}

Serial.println(currentPage);

displayPages(currentPage);

}

void displayPages(int currentPage){

int xValues[] = {0, 1, 2, 3, 4, 5};

int yValues[] = {10, 20, 15, 30, 25, 35};

int numPoints = sizeof(xValues) / sizeof(xValues[0]);

int myIntList[] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};

int listLength = sizeof(myIntList) / sizeof(myIntList[0]);

switch (currentPage){

case 0:

displayText("Page 0: Example Text");

break;

case 1:

displayText("Page 1: Example Nums: \n 1 1 2 ");

break;

case 2:

displayIntList(myIntList, listLength);

break;

case 3:

displayGraph(xValues, yValues, numPoints);

break;

default:

displayText("default");

break;

}

return;

}

void displayText(const char\* text) {

Serial.println("displaytext");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

display.setCursor(20, 50);

display.print(text);

display.update();

delay(1000);

}

void displayIntList(int list[], int length) {

Serial.println("Displaying Integer List");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

// Set initial cursor position

int x = 20;

int y = 50;

// Display each element of the list

for (int i = 0; i < length; i++) {

display.setCursor(x, y);

display.print(list[i]);

// Move the cursor to the next position

y += 20; // You can adjust the vertical spacing based on your preference

// Check if the next position goes beyond the display height

if (y > display.height() - 20) {

// If it does, reset y and move to the next column

y = 50;

x += 60; // You can adjust the horizontal spacing based on your preference

}

}

display.update();

delay(1000);

}

void displayGraph(int xValues[], int yValues[], int numPoints) {

Serial.println("Displaying Graph");

display.fillScreen(GxEPD\_WHITE); // Clear the screen

display.setTextColor(GxEPD\_BLACK);

display.setFont(&FreeMonoBold9pt7b);

// Define graph properties

int xStart = 20; // Starting X-coordinate of the graph

int yStart = 20; // Starting Y-coordinate of the graph

int graphWidth = 200; // Width of the graph

int graphHeight = 100; // Height of the graph

// Calculate the X and Y scale factors

float xScale = graphWidth / (float)(numPoints - 1);

float yScale = graphHeight / (float)(\*std::max\_element(yValues, yValues + numPoints) - \*std::min\_element(yValues, yValues + numPoints));

// Draw the X and Y axes

display.drawLine(xStart, yStart + graphHeight, xStart + graphWidth, yStart + graphHeight, GxEPD\_BLACK); // X-axis

display.drawLine(xStart, yStart, xStart, yStart + graphHeight, GxEPD\_BLACK); // Y-axis

// Draw the graph points and connecting lines

for (int i = 0; i < numPoints - 1; i++) {

int x1 = xStart + i \* xScale;

int y1 = yStart + graphHeight - ((yValues[i] - \*std::min\_element(yValues, yValues + numPoints)) \* yScale);

int x2 = xStart + (i + 1) \* xScale;

int y2 = yStart + graphHeight - ((yValues[i + 1] - \*std::min\_element(yValues, yValues + numPoints)) \* yScale);

// Draw connecting lines

display.drawLine(x1, y1, x2, y2, GxEPD\_BLACK);

// Draw circles at each data point

display.fillCircle(x1, y1, 2, GxEPD\_BLACK);

display.fillCircle(x2, y2, 2, GxEPD\_BLACK);

}

display.update();

delay(1000);

}

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// GENERAL FUNCTIONS

void setup() {

// set up serial monitor

Serial.begin(9600);

Serial.print("hey girl hey");

// set up color sensor

tcs.begin();

Wire.begin(SDA\_PIN, SCL\_PIN);

if (tcs.begin()) {

Serial.println("Found sensor");

} else {

Serial.println("No TCS34725 found ... check your connections");

while (1);

}

// set up temperature sensor

sensors.begin();

// set up button

pinMode(BUTTON\_PIN, INPUT\_PULLUP);

attachInterrupt(digitalPinToInterrupt(BUTTON\_PIN), buttonISR, FALLING);

// set up screen

display.init();

display.setRotation(1);

// decide on units of temperature

chooseFahrenheitCelsius();

// run first round of tests

block2();

}

void loop() {

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// POTENTIOMETER STUFF

//Read the potentiometer value

currentPotValue = analogRead(POTENTIOMETER\_PIN);

// Check if the potentiometer value has changed significantly

if (abs(currentPotValue - lastPotValue) > BUTTON\_THRESHOLD) {

Serial.print("Potentiometer value: ");

Serial.println(currentPotValue);

// Add code here to handle potentiometer value change (e.g., adjust brightness)

}

lastPotValue = currentPotValue;

bool leftRight = leftRightFunc(currentPotValue);

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// BUTTON STUFF

currentButtonState = digitalRead(BUTTON\_PIN);

if (lastButtonState == LOW && currentButtonState == HIGH) {

Serial.println("Button pressed");

switchPages(leftRight);

// Add code here to handle button press (e.g., switch pages)

}

lastButtonState = currentButtonState;

Figure 3: User Interface Arduino Code

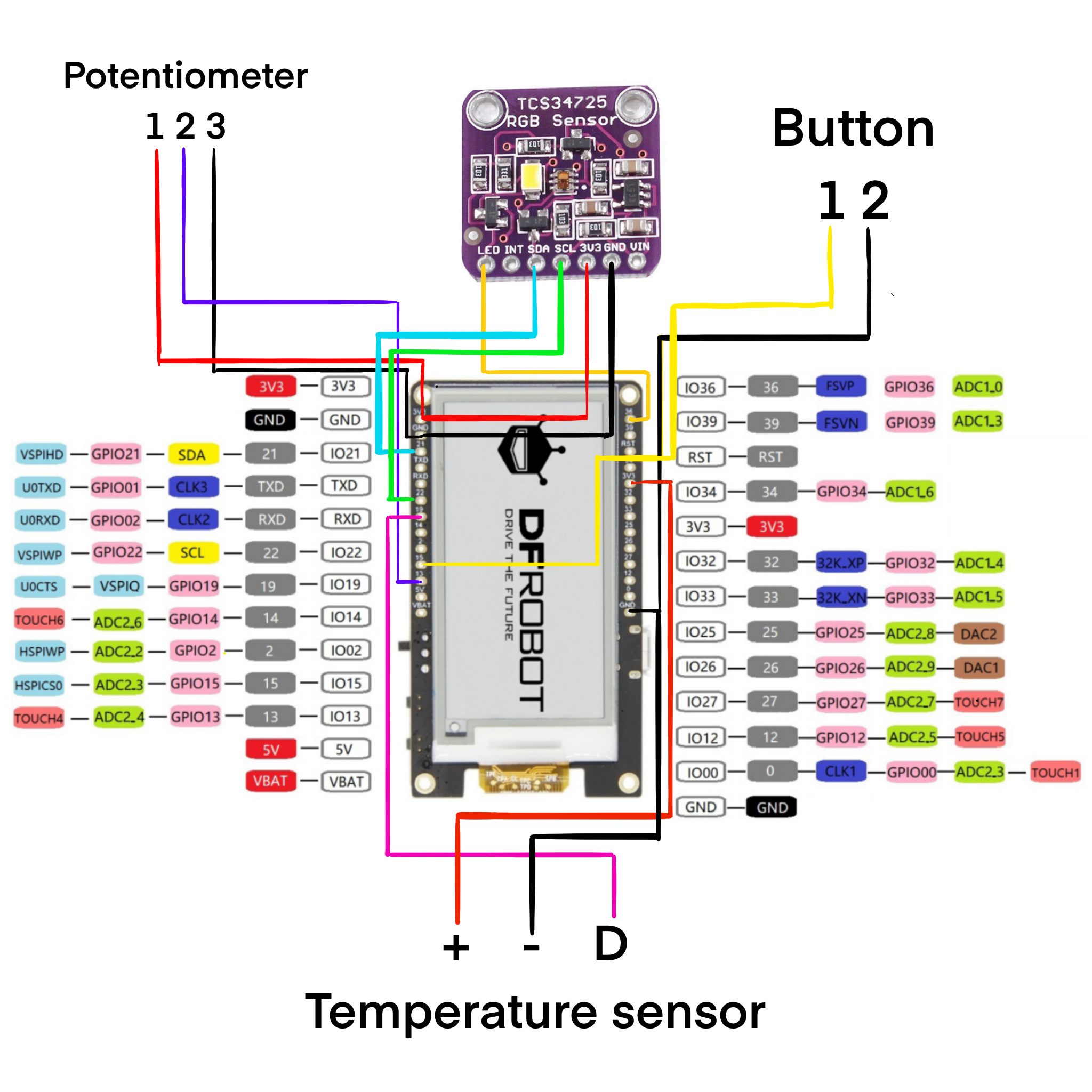


Figure 4: Wiring Diagram

TABLE I

User Interface Usability Results

| Subject | Following your review of the provided UI demonstration, would you agree that the output is useful? | Following your review of the provided UI demonstration, would you agree that the output is usable? | Following your review of the provided UI demonstration, would you agree that the output is understandable? | Provide any comments you have below: |
| --- | --- | --- | --- | --- |
| 1 | Yes | Yes | Yes |  |
| 2 | Yes | Yes | I wish it had 24 hour time but I understood it still |  |
| 3 | No | Yes | Yes | I don't have fish |
| 4 | Yes | Yes | Yes |  |
| 5 | Yes | Yes | Yes | Does it work with salt water too? I couldn't tell |
| 6 | Yes | Yes | Yes |  |
| 7 | Yes | Yes | Yes |  |
| 8 | Yes | Yes | Yes |  |
| 9 | Yes | Yes | Yes |  |
| 10 | Yes | Yes | Yes |  |

## 3. General Validation

The design of the user interface block for the fish tank monitor has been chosen to align with the specific needs and objectives of the system. Several factors were taken into consideration including cost-effectiveness, part availability, user friendliness, performance, interaction between blocks, size constraints, and the needs of potential users.

Firstly, the choice of a rectangular e-ink display was made based on a combination of its cost-effectiveness and energy efficiency. E-ink displays only update upon request, reducing power consumption and ensuring a longer lifespan for the monitor [1]. As an alternate solution, a different display technology, such as an LCD screen, could have been considered, however the choice of e-ink aligns with the project's focus on energy efficiency and simplicity, catering to the needs of both users and the overall system.

The simplicity and easy understanding of the layout were prioritized to increase accessibility for all users. The absence of color in the design, while in part due to limitations of the e-ink screen, contributes to a clean and straightforward visual representation. This design decision not only enhances user experience but also reduces potential distractions, keeping the device's sole focus on aquatic care.

Due to the interconnected nature of the system, the user interface is designed to provide a seamless experience across pages. The cohesive design ensures that users can easily navigate between functionalities, contributing to user-friendly interactions.

The user interface block is designed with the purpose of connecting the user's intentions with the technology. Since the user is able to interact with the user interface through the use of buttons and a potentiometer, they are able to move between pages on the screen. These different pages serve different purposes which each allow the user to commit some sort of action. These actions all serve a role in maintaining the health of an aquarium, contributing to the goal of the project. The details of the user interface block fit the system's need because the user interface is how the user is connecting with the system.

In conclusion, the thoughtful design of the user interface block takes into account various considerations to meet the specific requirements of the fish tank monitor system. By balancing factors like cost, accessibility, customization, and long-term monitoring, the user interface contributes significantly to the overall success of the fish tank monitoring system.

TABLE II

USER INTERFACE BLOCK INTERFACE VALIDATION

| **Interface Property** | **Why is this interface this value?** | **Why do you know that your design details for this block**  **above meet or exceed each property?** |
| --- | --- | --- |
| **otsd\_usr\_ntrfc\_usrin : Input** | | |
| Other: interrupt based | We want the button to be operational at all times and not dependent on a loop | Interrupt is attached to the button pin via code in Figure 3 line 511. |
| Type: multi-directional scrolling with potentiometer | We want the user to be able to scroll between pages in either direction for better usability | The "leftRightFunc" function returns a bool representing which direction the user is scrolling based on the potentiometer value (Figure 3, line 341); this boolean is passed to the "switchPages" (Figure 3, line 354) to determine which screen to scroll to upon a button push (Figure 3, lines 543-9). |
| Type: button pushes | The user needs to be able to move between screens and the button is a simple way for the user to indicate that they'd like to scroll. | The "switchPages" function is called immediately upon a button push to change the screen (Figure 3, lines 543-9) |
| **usr\_ntrfc\_scrn\_usrout : Output** | | |
| Other: paper-like | Paper-like screens do not change their displays until updated and we wanted to choose a screen that used less power. | The screen is a paper-like screen [1] |
| Type: Screen output consists of graphs, lists, numbers, and text (all of which are depicting some information about the aquarium's water parameters) | To monitor a fish tank, the user must be able to see numerical and text values to represent their tank's parameters, and the graphs and lists are effective ways of displaying that data over time. | The "displayText" function displays text and numbers on screen (Figure 3, line 399-407), the "displayIntList" displays a list of integers on screen (Figure 3, line 411-39), and "displayGraph" displays a graph on screen (Figure 3, line 441-83). |
| Usability: Output must be useful and usable, must be understandable by 9/10 users | The product's outputs must be considered useful and usable by a majority of users in order to have purpose and justify its design. | I demonstrated the user interface [(UI demonstration)](https://docs.google.com/document/d/1SDx4zR7Ocvez8J7AejJ-t2Y9T742h6ok8v9GinSnJfE/edit?usp=sharing) to 10 anonymous subjects (none of whom are members of the Oregon State University college of Engineering) and collected their results (Table I). |
| **usr\_ntrfc\_scrn\_data : Output** | | |
| Datarate: minimum of 1 measurement/second | The system needs to be able to obtain data from the system frequently to maintain up-to-date parameter information | The thermometer's stated NV Write Cycle Time is typically 2ms and has a max of 10ms [2] |
| Messages: user interface tells screen what to display depending on user input through buttons. Displays screens with graphs, lists, text, and numbers. | The user's selections through the button should inform the system of next steps requested by the user. | The "switchPages" function is called immediately upon a button push to change the screen (Figure 3, lines 542-549). The various screens can display graphs, lists, text, and numbers through the use of the "displayText", "displayIntList", and "displayGraph" functions (Figure 3, lines 378-94) |
| Protocol: I2C | The I2C protocol supports bi-directional communication, providing a reliable and standardized communication method for the sensor control block and the main board | Both the color sensor datasheet [3] and the screen datasheet [1] detail their compatibility with I2C. The I2C communication is initialized with the line "Wire.begin(SDA\_PIN, SCL\_PIN)" (Figure 3, Line 497). |

## 5. Verification Plan

1. Testing Interrupt-Based Input (otsd\_usr\_ntrfc\_usrin)
   1. Simulate user interactions triggering interrupts (button presses)
   2. Verify that the interrupt-based design responds promptly and efficiently
   3. Ensure that the system does not constantly check for conditions, validating the interrupt-based approach
2. Testing Multi-Directional Input (otsd\_usr\_ntrfc\_usrin)
   1. Use buttons in various directions to navigate through the system
   2. Confirm that the multi-directional input enhances the user's interaction
   3. Ensure flexibility in navigating different functionalities
3. Testing Button-Based Navigation (otsd\_usr\_ntrfc\_usrin)
   1. Simulate button presses to navigate between pages and functionalities
   2. Validate that the button-based navigation is intuitive and easily understood
   3. Confirm practicality and effectiveness of button-based input
4. Testing "Paper-Like" Screen Output (usr\_ntrfc\_scrn\_usrout)
   1. Display various types of content on the e-ink screen (graphs, lists, numbers, text, and illustrations)
   2. Verify that the screen mimics the appearance of ink on paper
   3. Confirm that the screen only uses power when the content changes
5. Testing Screen Output Usability (usr\_ntrfc\_scrn\_usrout)
   1. Observe and collect feedback from users with varying levels of experience
   2. Ensure that the output is useful and understandable by at least 9 out of 10 users
   3. Validate that the graphical and list representations meet usability standards
6. Testing Datarate Compatibility (usr\_ntrfc\_scrn\_data)
   1. Send data at the specified rate (1 measurement/second) to verify the datarate
   2. Verify that the datarate suits the processing speed of the ESP32
   3. Ensure optimal communication without data transfer bottlenecks
7. Testing User Interface Messages (usr\_ntrfc\_scrn\_data)
   1. Send messages from the user interface to the screen to update pages
   2. Confirm that the screen responds correctly to user inputs with the appropriate updates
   3. Validate that messages instruct the screen to display graphs, lists, illustrations, text, and numbers
8. Testing I2C Protocol (usr\_ntrfc\_scrn\_data)
   1. Implement bi-directional communication using the I2C protocol
   2. Verify that the communication is reliable and consistent
   3. Ensure that I2C protocol supports the standardized communication required for the system
9. Comprehensive Integration Testing
   1. Integrate the user interface block into the overall fish tank monitor system
   2. Test the block's interaction with other blocks and components
   3. Validate that the user interface seamlessly contributes to the effective management of the fish tank environment
10. User Feedback and Iterative Testing
    1. Gather feedback from users during real-world usage
    2. Iterate on any identified issues or areas for improvement
    3. Conduct additional testing if changes are made to the user interface block
11. Validation against System Requirements
    1. Verify that the user interface block, when integrated into the system, meets all specified system requirements
    2. Ensure that the block contributes significantly to the overall success of the fish tank monitoring system

## 6. References and File Links

### 6.1 References

[1] “2.13"e-ink Display module with ESP32 MCU wiki,” DFRobot, https://wiki.dfrobot.com/e-Ink\_Display\_Module\_for\_ESP32\_SKU\_DFR0676 (accessed Mar. 10, 2024).

[2] Maxim Integrated, "Programmable Resolution 1-Wire Digital Thermometer," DS18B20 datasheet, Revised Aug. 2019

[3] Texas Advanced Optoelectronic Solutions, "Color Light-To-Digital Converter with IR Filter," TCS3472 datasheet, Aug. 2012

### 6.2 File Links

[User Interface Paper Prototype](https://docs.google.com/document/d/1qcesZ8mIA9sEpSjqNkljarwgtFWUU8wRginwR6MpLrY/edit?usp=sharing)

[Interactive UI (slides)](https://docs.google.com/presentation/d/1fcd7UCvtgRXrfySkEQBxXgiFbmlICC6Yp_s5Ec1n-Jc/edit?usp=sharing)

[UI Demonstration Instructions](https://docs.google.com/document/d/1SDx4zR7Ocvez8J7AejJ-t2Y9T742h6ok8v9GinSnJfE/edit?usp=sharing)

#### 6.2.1 Datasheet PDFs

[Color Sensor](https://drive.google.com/file/d/11KFGTJVkesoG8jHHQOEfUxUZ5dy5fkVU/view?usp=drive_link)

[Screen](https://drive.google.com/file/d/1LCnIXtZZ29pK2zgxB5DYRpUhkZnG7cQx/view?usp=drive_link)

[Thermometer](https://drive.google.com/file/d/1Wr788dnIJMPOlnrYm890tkVMwFFs2bmD/view?usp=drive_link)

## 7. Revision Table

| **Date** | **Action** |
| --- | --- |
| 1/24/2024 | Savannah Tanner   * document creation; sections 1-7 filled in; section completed; |
| 3/7/2024 | Savannah Tanner   * Updated interfaces * Revised based on instructor comments from draft |
| 3/10/2024 | Savannah Tanner   * Finalized edits * Added citations * Added wiring diagram * Added revision statement * Updated formatting * Added code * Added table 1 and updated table 2 |

## 8. Revision Statement

Since the initial draft of this document there have been many changes and alterations made to the User Interface Block. The block changed from a user-interface to a code block. This allowed for the voltage and current interface requirements to be removed. Several interface properties were altered such as the usr\_ntrfc\_scrn\_data Datarate property. This changed from 400 kilobytes to a value of 1 measurement/second. Measurements per second is a more appropriate unit given the needs of the system and it is also easier to monitor and evaluate. I also added two new figures to the document as well as adjusted the name for one. I renamed Figure 2 to "User Interface State Diagram" as the diagram only shows what is visible to the user, while a flow chart would start from the initialization. I added Figure 3, "User Interface Arduino Code", which contains all the functions necessary to display the user interface, and I also added Figure 4, "Wiring Diagram", which is the most up to date wiring diagram needed to operate the user interface without the team PCB. Lastly, I made revisions based on draft feedback. These changes include: updating the description to provide a more direct and explicit statement on the block's functions, ensuring my language was technical and precise throughout, correcting the general validation section to refer to elements of section two, adjusting the interface validation table to follow the provided format, fixing the justification for the design details, and updating my references and file links.