

Embedded Systems Summative

0.1

Generated by Doxygen 1.10.0

1 File Index	1
1.1 File List	1
2 File Documentation	3
2.1 BUZZER.c	3
2.2 buzzer.h	3
2.3 GLCDTOUCH.c	4
2.4 GLCDTOUCH.h	9
2.5 interrupt.c	10
2.6 interrupt.h	11
2.7 keypad.c	11
2.8 keypad.h	14
2.9 led_main.c	14
2.10 led_main.h	15
2.11 photoresistor.c	15
2.12 photoresistor.h	17
2.13 Safe.c	17
2.14 screens.c	18
2.15 screens.h	20
2.16 servo.c	20
2.17 servo.h	21
2.18 shock.c	22
2.19 shock.h	22
2.20 tracking.c	23
2.21 tracking.h	23
Index	25

Chapter 1

File Index

1.1 File List

Here is a list of all documented files with brief descriptions:

EmbeddedSummative/ BUZZER.c	3
EmbeddedSummative/ buzzer.h	3
EmbeddedSummative/ GLCDTOUCH.c	4
EmbeddedSummative/ GLCDTOUCH.h	9
EmbeddedSummative/ interrupt.c	10
EmbeddedSummative/ interrupt.h	11
EmbeddedSummative/ keypad.c	11
EmbeddedSummative/ keypad.h	14
EmbeddedSummative/ led_main.c	14
EmbeddedSummative/ led_main.h	15
EmbeddedSummative/ photoresistor.c	15
EmbeddedSummative/ photoresistor.h	17
EmbeddedSummative/ Safe.c	17
EmbeddedSummative/ screens.c	18
EmbeddedSummative/ screens.h	20
EmbeddedSummative/ servo.c	20
EmbeddedSummative/ servo.h	21
EmbeddedSummative/ shock.c	22
EmbeddedSummative/ shock.h	22
EmbeddedSummative/ tracking.c	23
EmbeddedSummative/ tracking.h	23

Chapter 2

File Documentation

2.1 BUZZER.c

```
00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     BUZZER.c
00010
00011     Description:
00012
00013     BUZZER.c initialises the buzzer, and includes functions for setting and resetting the pin for the
buzzer
00014
00015     #####
00016 */
00017
00018 #include <stdio.h>
00019 #include "stm32f7xx_hal.h"
00020 #include "stm32f7xx_hal_gpio.h"
00021 #include "buzzer.h"
00022
00023 GPIO_InitTypeDef gpio_buzz;
00024
00025 // Function to initialise the buzzer
00026 void init_buzzer(){
00027     __HAL_RCC_GPIOH_CLK_ENABLE(); // GPIO port H
00028
00029     gpio_buzz.Mode = GPIO_MODE_OUTPUT_PP;
00030     gpio_buzz.Pull = GPIO_NOPULL;
00031     gpio_buzz.Speed = GPIO_SPEED_HIGH;
00032     gpio_buzz.Pin = GPIO_PIN_6; // Pin 6
00033
00034     HAL_GPIO_Init(GPIOH, &gpio_buzz); // Initialise the pin
00035
00036 }
00037 // Function to turn on the buzzer
00038 void buzz_on(){
00039     HAL_GPIO_WritePin(GPIOH, gpio_buzz.Pin, GPIO_PIN_SET);
00040 }
00041
00042 // Function to turn off the buzzer
00043 void buzz_off(){
00044     HAL_GPIO_WritePin(GPIOH, gpio_buzz.Pin, GPIO_PIN_RESET);
00045 }
```

2.2 buzzer.h

```
00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
```

```

00006
00007     File name:
00008
00009     buzzer.h
00010
00011     Description:
00012
00013     buzzer.h is used to include the buzzer.c definitions inside other source files.
00014
00015     #####
00016     */
00017 void init_buzzer();
00018 void buzz_off();
00019 void buzz_on();

```

2.3 GLCDTOUCH.c

```

00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     GLCDTOUCH.C
00010
00011     Description:
00012
00013     GLCDTOUCH.C handles logic surrounding unlocking, resetting and setting the pin for the system. It
    also handles transitions between screens defined in Screens.c.
00014     GLCDTOUCH.C also handles 2FA for the UART/USART bluetooth communication.
00015     #####
00016     */
00017     #include "stm32f7xx_hal.h"
00018     #include "GLCD_Config.h"
00019     #include "Board_GLCD.h"
00020     #include "Board_Touch.h"
00021     #include <string.h>
00022     #include "buzzer.h"
00023     #include "keypad.h"
00024     #include "screens.h"
00025     #include "led_main.h"
00026     #include "servo.h"
00027     #include "tracking.h"
00028     #include "led_main.h"
00029     #include "interrupt.h"
00030     #define wait_delay HAL_Delay
00031     extern GLCD_FONT GLCD_Font_16x24; // Externally including font for GLCD text
00032     int pin[8] = {1,2,3,4,5,6,7,8}; // Pin array that holds up to 8 integers, used for multiple
    functions. Preset for testing purposes.
00033     char status[6] = "Locked"; // Current status for the system, set to locked and is
    displayed on the home screen
00034     int attempts = 0; // Records the amount of attempts the user has made to enter
    the currently set pin
00035     extern int ScreenState; // Externally included variable from Screens.c, this
    variable is used to check the current screen the user is interacting with
00036     TOUCH_STATE tsc_state; // Stores the current TOUCH_STATE
00037     UART_HandleTypeDef Bluetooth_UARTHandle; // Handle for enabling UART bluetooth communication
    with HC-05
00038     uint8_t RxBuffer; // Buffer for holding received data from bluetooth
    communication
00039
00040
00041 #ifdef __RTX
00042 extern uint32_t os_time;
00043 uint32_t HAL_GetTick(void) {
00044     return os_time;
00045 }
00046 #endif
00047 // Function for initialising UART, it initialises RX and TX ports
00048 void init_uart(void){ // Initialising UART capabilities
00049     GPIO_InitTypeDef GPIO_InitStructure;
00050
00051     // Initialising RX port for D0
00052     GPIO_InitStructure.Pin = GPIO_PIN_7; // Pin 7
00053     GPIO_InitStructure.Mode = GPIO_MODE_AF_PP; // Push Pull config
00054     GPIO_InitStructure.Alternate = GPIO_AF8_USART6; // Initialising USART6 as it is mapped to pin D0
    on board
00055     GPIO_InitStructure.Speed = GPIO_SPEED_HIGH;
00056     GPIO_InitStructure.Pull = GPIO_NOPULL;
00057     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure); // Initialising the pin on port C
00058     // Initialising TX port for D1

```



```

00059     GPIO_InitStructure.Pin = GPIO_PIN_6;           // PIN 6
00060     GPIO_InitStructure.Mode = GPIO_MODE_AF_PP;      // Push Pull config
00061     HAL_GPIO_Init(GPIOC, &GPIO_InitStructure);     // Initialising the pin on port C
00062     // Configuring the bluetooth handle
00063     Bluetooth_UARTHandle.Instance = USART6;
00064     Bluetooth_UARTHandle.Init.BaudRate = 9600;      // Setting BaudRate to 9600, this is because the
chip I bought online recommends/requires 9600
00065     Bluetooth_UARTHandle.Init.WordLength = UART_WORDLENGTH_8B;
00066     Bluetooth_UARTHandle.Init.StopBits = UART_STOPBITS_1;
00067     Bluetooth_UARTHandle.Init.Parity = UART_PARITY_NONE;
00068     Bluetooth_UARTHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00069     Bluetooth_UARTHandle.Init.Mode = UART_MODE_TX_RX;
00070     HAL_UART_Init(&Bluetooth_UARTHandle);          // Initialising the handle
00071 }
00072
00073
00074 void SystemClock_Config(void) { // Configuring System Clock
00075     RCC_OscInitTypeDef RCC_OscInitStruct;
00076     RCC_ClkInitTypeDef RCC_ClkInitStruct;
00077     /* Enable Power Control clock */
00078     __HAL_RCC_PWR_CLK_ENABLE();
00079     /* The voltage scaling allows optimizing the power
00080     consumption when the device is clocked below the
00081     maximum system frequency. */
00082     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
00083     /* Enable HSE Oscillator and activate PLL
00084     with HSE as source */
00085     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00086     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00087     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00088     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00089     RCC_OscInitStruct.PLL.PLLM = 25;
00090     RCC_OscInitStruct.PLL.PLLN = 336;
00091     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00092     RCC_OscInitStruct.PLL.PLLQ = 7;
00093     HAL_RCC_OscConfig(&RCC_OscInitStruct);
00094     /* Select PLL as system clock source and configure
00095     the HCLK, PCLK1 and PCLK2 clocks dividers */
00096     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK |
RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00097     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00098     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00099     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
00100     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
00101     HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
00102 }
00103
00104
00105 // Function to delay the system for 1 second.
00106 void delay(){
00107     HAL_Delay(1000);
00108 }
00109
00110 // 2FA verification method, allows user to respond on compatible bluetooth device to confirm unlock
function
00111 bool verify(){
00112     init_uart(); // Initialising uart
00113     RxBuffer = 0; // Resetting RxBuffer each time to avoid potential issues
00114     GLCD_DrawString(56, 100, "Please confirm on device.");
00115
00116     HAL_UART_Transmit(&Bluetooth_UARTHandle, (uint8_t *) "Confirm", 7, HAL_MAX_DELAY); // Prints
"Confirm" on connected bluetooth terminal
00117     HAL_UART_Receive(&Bluetooth_UARTHandle, &RxBuffer, 1, HAL_MAX_DELAY); // Takes
response from bluetooth terminal and stores in RxBuffer
00118
00119
00120     if (RxBuffer != 'y'){ // If the buffer is not equal to 'y'
00121         return false; // Verification failed
00122     }
00123     // If verified
00124     buzz_on(); // buzzer on
00125     delay(); // prolong buzzing
00126     buzz_off(); // turn buzzer off
00127     return true; // return successful
00128 }
00129 }
00130 // Unlock function, checks a temporary array filled with Membrane Keypad inputs and compares to the
current stored pin
00131 bool unlock(){
00132     int attempt[8]; // attempt array, this stores the attempt that the user will input on the
membrane keypad
00133     int i = 0; // Loop variable which is incremented
00134     int spacing = 37; // This is used to space the asteriks inline with the drawn _'s.
00135     while(i<8){
00136         int Membrane = getInput(); // Getting input from the membrane keypad
00137         if (ScreenState != 2){ // If its not the unlock screen, break. The reason being is
because this function should not run on any other screens
00138             break;

```

```

00139     }
00140
00141     Touch_GetState(&tsc_state); // Getting the touch state
00142     if(tsc_state.pressed){
00143         CheckCoords(tsc_state.x, tsc_state.y); // If CheckCoords is true, the ScreenState
will change and the function will break based on the above if conditional
00144         tsc_state.pressed = 0; // I'm resetting the touch state here,
just to avoid potential errors further into the program
00145     }
00146
00147     if(Membrane!=-1){ // If the membrane detects a valid input
00148         spacing+=42;
00149         GLCD_DrawString(spacing, 150, "*"); // Draw '*' for each input
00150         attempt[i] = Membrane; // store the input into index i of temporary array attempt
00151         i++; // increment
00152         delay(); // delay inputs using the delay function, prevents rapid
inputs from one press of button
00153     }
00154 }
00155
00156
00157 for (i = 0; i < 8; i++){
00158     if (attempt[i] != pin[i]){ // Checking the array of inputs against current stored pin
00159         return false; // If one attempt is wrong, return false. Makes it more
efficient as the worst case time complexity is O(n)).
00160     }
00161 }
00162 if (verify() != true){ // If the unlock is not verified return false
00163     return false;
00164 }
00165 return true; // If all loops continue, return true
00166 }
00167 }
00168 /*
00169  * Confirm function, this function takes in a temporary attempt array from a previous function and
checks for discrepancies between the two arrays.
00170  *
00171  */
00172 bool confirm(int attempt[]){
00173     int temp_pin[8]; // Temporary pin array to store confirmation attempt
00174     int i = 0; // Int that will be used to increment loops
00175     int spacing = 37; // Used for spacing asteriks that will be printed each input
00176
00177     while(i<8){
00178         int Membrane = getInput(); // Getting keypad input
00179         Touch_GetState(&tsc_state); // Getting the current touch state
00180
00181         if(tsc_state.pressed){
00182             CheckCoords(tsc_state.x, tsc_state.y); // CheckCoords to check for back button press
00183             tsc_state.pressed = 0; // reset tsc_state to avoid potential errors
00184         }
00185
00186         if (ScreenState != 5 && ScreenState != 6){ // If the screen is not on the confirmation
screens, break
00187             break;
00188         }
00189
00190         if(Membrane!=-1){ // If valid Membrane Keypad input
00191             spacing+=42;
00192             GLCD_DrawString(spacing, 150, "*");
00193             temp_pin[i] = Membrane; // store the current membrane input into temporary array
00194             i++; // increment i
00195             delay(); // delay
00196         }
00197     }
00198
00199     for (i =0; i < 8; i++){
00200         if (attempt[i] != temp_pin[i]){ // Checking the inputs against the array passed into
the confirm function
00201             return false; // returns false if any input does not equal the array
elements passed in the function
00202         }
00203     }
00204     return true; // If all loops finish, return true
00205 }
00206
00207 bool resetSetPin(){
00208     int temp_pin[8]; // Storing a temporary input
00209     int i = 0; // Int to increment loops
00210     int spacing = 37; // Int used to apply spacing to asteriks inputted
00211
00212     while(i<8){
00213         int Membrane = getInput();
00214
00215         if (ScreenState != 4){ // If the user is not on the resetSetScreen
00216             return false;
00217         }

```

```

00218     Touch_GetState(&tsc_state);
00219
00220     if(tsc_state.pressed){
00221         CheckCoords(tsc_state.x, tsc_state.y);
00222         tsc_state.pressed = 0;
00223     }
00224 }
00225 if(Membrane!=-1){
00226     spacing+=42;
00227     GLCD_DrawString(spacing, 150, "*");
00228     temp_pin[i] = Membrane;
00229     i++;
00230     delay();
00231 }
00232 }
00233
00234 GLCD_ClearScreen(); // Clearing the screen for the next screen
00235 ScreenState = 5; // Setting the ScreenState to be the ScreenState of the confirm screen,
essential for not immediately returning false
00236 confirmScreen();
00237
00238 while (ScreenState == 5){ // While user is on the confirm screen
00239     if (confirm(temp_pin)){ // If confirm returns true
00240         for (i = 0; i < 8; i++){
00241             pin[i] = temp_pin[i]; // Store new pin into pin global variable
00242         }
00243         return true; // return true
00244     } else {
00245         if (ScreenState != 5){ // If not on the confirm screen, return false
00246             return false;
00247         }
00248         GLCD_ClearScreen(); // Clear the screen
00249         confirmScreen(); // Go back to the confirm screen, this will also call the confirm
function again
00250         GLCD_DrawString(72, 100, "Incorrect, try again."); // Giving the user feedback
00251     }
00252 }
00253 }
00254
00255 bool setPin(){
00256     int temp_pin[8]; // Temporary pin to store input
00257     int i = 0;
00258     int spacing = 37;
00259
00260     while(i<8){
00261         int Membrane = getInput();
00262
00263         Touch_GetState(&tsc_state);
00264
00265         if(tsc_state.pressed){
00266             CheckCoords(tsc_state.x, tsc_state.y);
00267             tsc_state.pressed = 0;
00268         }
00269
00270         if(Membrane!=-1){
00271             spacing+=42;
00272             GLCD_DrawString(spacing, 150, "*");
00273
00274             temp_pin[i] = Membrane; // storing the inputs into the temporary array
00275             i++; // Incrementing loop
00276             delay();
00277         }
00278     }
00279
00280     GLCD_ClearScreen(); // Clearing the screen
00281     ScreenState = 6; // Setting the screen to the confirm screen for setting a new pin
00282     setPinConfirmScreen();
00283     while (ScreenState == 6){ // avoids the function from carrying over to the next screen
00284         if (confirm(temp_pin)){
00285             for (i = 0; i < 8; i++){
00286                 pin[i] = temp_pin[i]; // if confirmed, store in global pin array
00287             }
00288             return true;
00289         } else {
00290             if (ScreenState != 6){ // If not on correct screen, return false
00291                 return false;
00292             }
00293             GLCD_ClearScreen(); // Clearing the screen, this gets rid of previous asteriks
00294             setPinConfirmScreen(); // Go to the same screen
00295             GLCD_DrawString(72, 100, "Incorrect, try again."); // User feedback
00296         }
00297     }
00298 }
00299
00300 bool resetPin(){
00301     int attempt[8];
00302     int i = 0;

```

```

00303     int spacing = 37;
00304
00305     while(i<8){
00306         int Membrane = getInput();
00307         if (ScreenState != 3){          // If not on the reset screen, return false
00308             break;
00309         }
00310
00311         Touch_GetState(&tsc_state);
00312         if(tsc_state.pressed){
00313             CheckCoords(tsc_state.x, tsc_state.y);
00314             tsc_state.pressed = 0;
00315         }
00316
00317         if(Membrane!=-1){
00318             spacing+=42;
00319             GLCD_DrawString(spacing, 150, "*");
00320             attempt[i] = Membrane;
00321             i++;
00322             delay();
00323         }
00324     }
00325
00326     for (i = 0; i < 8; i++){
00327         if (attempt[i] != pin[i]){
00328             return false;
00329         }
00330     }
00331
00332     return true;
00333 }
00334
00335 // This function will reset the attempts global variable attempts and return the value. (Returning the
// value seemed to work whereas if I did not the return the value it did not work.)
00336 int attemptsReset(){
00337     attempts = 0;
00338     return attempts;
00339 }
00340 // Main function for changing between screens and calling the relevant functions for those screens
00341 void changeScreen(int a){
00342
00343     GLCD_SetFont(&GLCD_Font_16x24); // Setting the font for all the text
00344     photoresistor_main();           // Calls function from photoresistor.c. This function controls
// dark mode vs light mode
00345     GLCD_ClearScreen();              // Clearing the screen
00346     delay();                         // Delaying, reduces screen flickering
00347
00348     switch(a){
00349         case 0:                      // a = 0
00350             ScreenState = 0;         // ScreenState = 0, which is representative of the setPin screen
00351             setPinScreen();
00352
00353             while (ScreenState == 0 || ScreenState == 6){ // While at the setPinScreen or the
// setPinConfirm screen
00354                 if (setPin()){       // Calling the setPin function
00355                     changeScreen(1); // if it returns true, change to homescreen
00356                 }
00357                 break;
00358             case 1:
00359                 ScreenState = 1;     // Represents home screen
00360                 hscreen();           // home screen
00361                 GLCD_DrawString(192, 0*24, "Locked"); // Printing status for home screen
00362                 break;
00363             case 2:
00364                 ScreenState = 2;     // Represents unlock screen
00365                 unlockScreen();
00366                 attemptsReset();     // Resetting the attempts at the start, this prevents attempts
// carrying over from other screens or previous attempts
00367                 while (ScreenState == 2){
00368                     GLCD_ClearScreen();
00369                     unlockScreen();  // This is essential, as it will reset asteriks from previous
// attempts
00370                     attempts++;
00371                     timed_delay(attempts); // timed_delay is defined in interrupt.c
00372                     if (unlock()){    // if the unlock function returns true
00373                         changeScreen(4); // change to unlocked screen
00374                     }
00375                 }
00376                 break;
00377             case 3:
00378                 ScreenState = 3;     // Represents reset Screen
00379                 attemptsReset();     // Resetting attempts to avoid carry over
00380             }
00381     }
00382 }

```

```

00385         resetScreen();
00386
00387         while (ScreenState == 3 || ScreenState == 4 || ScreenState==5){ // While on resetScreen
or any of the confirm screens associated with resetting pin
00388             GLCD_ClearScreen();
00389             resetScreen();
00390             attempts++;
00391             timed_delay(attempts);
00392
00393             if (resetPin()){
00394                 GLCD_ClearScreen();
00395                 ScreenState=4; // represents screen to set after resetting
00396
00397                 resetSetScreen(); // screen to set pin after resetting
00398                 if(resetSetPin()) {
00399                     changeScreen(1); // return to home screen
00400                 }
00401             }
00402         }
00403     }
00404
00405     break;
00406
00407     case 4:
00408         ScreenState = 7; // Represents unlocked screen
00409         opened_angle(); // calls opened_angle() from servo.c, which turns servo by 90 degrees
00410         HAL_Delay(3000); // delay, allows time for user to open the door
00411         while (HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_3) == GPIO_PIN_SET){ // This turns out to be the
opposite, whenever the tracking sensor does not detect something, the pin for the sensor is set
00412             turnOnGreen(); // from led_main.c, turns led green
00413             lockedScreen();
00414             HAL_Delay(3000); // delay for 3 seconds
00415             if (HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_3) != GPIO_PIN_SET){ // If the tracking sensor
detects something, break out of while loop
00416                 break;
00417             }
00418         }
00419         closed_angle(); // servo.c function, turns servo 90 degrees opposite of what it was
just turned
00420         turnOnRed(); // turn on red led
00421         changeScreen(1); // go to home screen
00422         break;
00423
00424     default:
00425         break;
00426 }
00427 }
00428

```

2.4 GLCDTOUCH.h

```

00001
00002 /*
00003     #####
00004     Principle Author:
00005
00006     Tommy Peach, 100358179
00007
00008     File name:
00009     GLCDTOUCH.h
00010
00011
00012     Description:
00013
00014     GLCDTOUCH.h defines functions that will be used in other source files
00015
00016     #####
00017 */
00018 #include "stm32f7xx_hal.h"
00019 #include "GLCD_Config.h"
00020 #include "Board_GLCD.h"
00021 #include "Board_Touch.h"
00022 #include <string.h>
00023 #include "buzzer.h"
00024 #include "keypad.h"
00025 #include "led_main.h"
00026 #include "servo.h"
00027 #include "tracking.h"
00028 #include "interrupt.h"
00029 #define wait_delay HAL_Delay
00030
00031 extern GLCD_FONT GLCD_Font_16x24;
00032 extern UART_HandleTypeDef Bluetooth_UARTHandle;

```

```

00033     extern uint8_t RxBuffer;
00034     extern int pin[8];
00035     extern TOUCH_STATE tsc_state;
00036     extern int ScreenState;
00037     extern int attempts;
00038
00039
00040 void changeScreen();
00041 _Bool verify();
00042 void delay();
00043 void setPin();
00044 _Bool unlock();
00045 void init_uart(void);
00046 void SystemClock_Config(void);
00047 _Bool resetPin();
00048 _Bool confirm(int attempt[]);
00049 void setPin();

```

2.5 interrupt.c

```

00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     interrupt.c
00010
00011     Description:
00012
00013     interrupt.c handles delaying the system when too many incorrect attempts are entered. It also
00014     handles the interrupt caused by the shock sensor.
00015     #####
00016 */
00017
00018 #include <stdio.h>
00019 #include "stm32f7xx_hal.h"
00020 #include "stm32f7xx_hal_gpio.h"
00021 #include "Board_GLCD.h"
00022
00023 #include "buzzer.h"
00024 char countdown[1]; // buffer for showing seconds for delay
00025
00026 // Function for delaying and printing the number of seconds until countdown ends
00027 void delay_seconds(int seconds)
00028 {
00029     int i = 0;
00030     for (i = seconds; i >= 0; i--){           // This for loop decrements until the integer i
00031         reaches 0
00032         sprintf(countdown, "Delay: %d", i); // sprintf puts the text "Delay: %d" into the
00033         countdown buffer. %d is the identifier for what data type is meant to take its place.
00034         GLCD_DrawString(168,9*24, countdown); // Printing the countdown on the screen
00035         HAL_Delay(1000);                      // Delays the program for 1 second, 1000 because it is
00036         milliseconds (1000 milliseconds = 1 second)
00037     }
00038 }
00039
00040 // Function for checking the interrupt, the function checks for the interrupt and if there is one the
00041 // callback function will execute
00042 void interrupt_check(void)
00043 {
00044     // This function will check for the shock sensor interrupt, the gpio pin is gpio_pin_6 as this
00045     // is the interrupt handler vector I configured in shock.c
00046     HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
00047 }
00048
00049 // Callback function for the interrupt, this function writes the pin for the buzzer, uses a for loop
00050 // to delay the resetting of the buzzer pin
00051 void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin){
00052     HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_SET); // Setting the buzzer pin
00053     for (i = 0; i < 10000000; i++){                      // for loop to delay the reset
00054         of the buzzer, causing it to continue sounding
00055     }
00056     HAL_GPIO_WritePin(GPIOH, GPIO_PIN_6, GPIO_PIN_RESET); // resetting the buzzer
00057 }
00058
00059 // Function to delay the system, based on the amount of attempts have accumulated in GLCDTOUCH.c
00060 void timed_delay(int wrong_attempts){
00061

```

```

00056 // If two wrong attempts
00057 if (wrong_attempts == 2)
00058 {
00059     GLCD_DrawString(72, 100, "Incorrect, try again."); // User feedback
00060     delay_seconds(5); // Calls the delay_seconds function, wait for 5 seconds
00061 }
00062 else if (wrong_attempts == 3)
00063 {
00064     // Wait for 10 seconds
00065     GLCD_DrawString(72, 100, "Incorrect, try again.");
00066     delay_seconds(10);
00067 }
00068 else if (wrong_attempts >= 4)
00069 {
00070     // Wait for 15 seconds
00071     GLCD_DrawString(72, 100, "Incorrect, try again.");
00072     delay_seconds(15);
00073 }
00074 }
00075 }
00076 }
00077 }
00078 }
00079 }
00080 }
00081 }
00082 }
00083 }
00084 }

```

2.6 interrupt.h

```

00001 /*
00002     #####
00003     Principle Author:
00004     Tommy Peach, 100358179
00005     File name:
00006     interrupt.h
00007     Description:
00008     interrupt.h defines functions that will be used in other source files.
00009     #####
00010 */
00011 void timed_delay(int wrong_attempts);

```

2.7 keypad.c

```

00001 #include "keypad.h"
00002 #include "stm32f7xx_hal.h"
00003 #include "stm32f7xx_hal_gpio.h"
00004 void initializeMembranePins (void){
00005     GPIO_InitTypeDef gpio;
00006     __HAL_RCC_GPIOA_CLK_ENABLE();
00007     __HAL_RCC_GPIOI_CLK_ENABLE();
00008     __HAL_RCC_GPIOB_CLK_ENABLE();
00009     gpio.Mode = GPIO_MODE_OUTPUT_PP;
00010     gpio.Pull = GPIO_PULLDOWN;
00011     gpio.Speed = GPIO_SPEED_HIGH;
00012     // D8
00013     gpio.Pin = GPIO_PIN_2;
00014     HAL_GPIO_Init(GPIOI, &gpio);
00015     // D9
00016     gpio.Pin = GPIO_PIN_15;
00017     HAL_GPIO_Init(GPIOA, &gpio);
00018     // D10
00019     gpio.Pin = GPIO_PIN_8;
00020     HAL_GPIO_Init(GPIOA, &gpio);
00021     // D11

```

```

00029     gpio.Pin = GPIO_PIN_15;
00030     HAL_GPIO_Init(GPIOB, &gpio);
00031
00032     // D12
00033     gpio.Pin = GPIO_PIN_14;
00034     HAL_GPIO_Init(GPIOB, &gpio);
00035
00036     // D13
00037     gpio.Pin = GPIO_PIN_1;
00038     HAL_GPIO_Init(GPIOI, &gpio);
00039
00040     // D14
00041     gpio.Pin = GPIO_PIN_9;
00042     HAL_GPIO_Init(GPIOB, &gpio);
00043
00044     // D15
00045     gpio.Pin = GPIO_PIN_8;
00046     HAL_GPIO_Init(GPIOB, &gpio);
00047 }
00048
00049 int convertPinsToNum(int k, int r){
00050     if(k == 11 && r == 15){
00051         return 1;
00052     }
00053     if(k == 10 && r == 15){
00054         return 2;
00055     }
00056     if(k == 9 && r == 15){
00057         return 3;
00058     }
00059     if(k == 8 && r == 15){
00060         return 11; //A
00061     }
00062     if(k == 11 & r == 14){
00063         return 4;
00064     }
00065     if(k == 10 & r == 14){
00066         return 5;
00067     }
00068     if(k == 9 & r == 14){
00069         return 6;
00070     }
00071     if(k == 8 & r == 14){
00072         return 12; //B
00073     }
00074     if(k == 11 & r == 13){
00075         return 7;
00076     }
00077     if(k == 10 & r == 13){
00078         return 8;
00079     }
00080     if(k == 9 & r == 13){
00081         return 9;
00082     }
00083     if(k == 8 & r == 13){
00084         return 13; //C
00085     }
00086     if(k == 11 & r == 12){
00087         return 15; // *
00088     }
00089     if(k == 10 & r == 12){
00090         return 0;
00091     }
00092     if(k == 9 & r == 12){
00093         return 16; // #
00094     }
00095     if(k == 8 & r == 12){
00096         return 14; //D
00097     }
00098
00099     return 0;
00100 }
00101
00102 void turnOn(int pinNo){
00103     switch (pinNo){
00104         case 8:
00105             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, GPIO_PIN_RESET);
00106             break;
00107         case 9:
00108             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_RESET);
00109             break;
00110         case 10:
00111             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_RESET);
00112             break;
00113         case 11:
00114             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET);
00115             break;

```



```

00116         case 12:
00117             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_RESET);
00118             break;
00119         case 13:
00120             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_1, GPIO_PIN_RESET);
00121             break;
00122         case 14:
00123             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_RESET);
00124             break;
00125         case 15:
00126             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_RESET);
00127             break;
00128     }
00129 }
00130
00131 void turnOff(int pinNo){
00132     switch (pinNo){
00133         case 8:
00134             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_2, GPIO_PIN_SET);
00135             break;
00136         case 9:
00137             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_15, GPIO_PIN_SET);
00138             break;
00139         case 10:
00140             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, GPIO_PIN_SET);
00141             break;
00142         case 11:
00143             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_SET);
00144             break;
00145         case 12:
00146             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_14, GPIO_PIN_SET);
00147             break;
00148         case 13:
00149             HAL_GPIO_WritePin(GPIOI, GPIO_PIN_1, GPIO_PIN_SET);
00150             break;
00151         case 14:
00152             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_9, GPIO_PIN_SET);
00153             break;
00154         case 15:
00155             HAL_GPIO_WritePin(GPIOB, GPIO_PIN_8, GPIO_PIN_SET);
00156             break;
00157     }
00158 }
00159
00160 GPIO_PinState readPin(int number){
00161     switch (number){
00162         case 8:
00163             return HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_2);
00164         case 9:
00165             return HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_15);
00166         case 10:
00167             return HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_8);
00168         case 11:
00169             return HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_15);
00170         case 12:
00171             return HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_14);
00172         case 13:
00173             return HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_1);
00174         case 14:
00175             return HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_9);
00176         case 15:
00177             return HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_8);
00178         default:
00179             return GPIO_PIN_RESET;
00180     }
00181 }
00182
00183 void setColsIn(){
00184     //set columns to write
00185     GPIO_InitTypeDef gpio;
00186     gpio.Mode = GPIO_MODE_INPUT;
00187     gpio.Pull = GPIO_PULLDOWN; //
00188     gpio.Speed = GPIO_SPEED_HIGH; //
00189     // D8
00190     gpio.Pin = GPIO_PIN_2;
00191     HAL_GPIO_Init(GPIOI, &gpio);
00192
00193     // D9
00194     gpio.Pin = GPIO_PIN_15;
00195     HAL_GPIO_Init(GPIOA, &gpio);
00196
00197     // D10
00198     gpio.Pin = GPIO_PIN_8;
00199     HAL_GPIO_Init(GPIOA, &gpio);
00200
00201     // D11
00202     gpio.Pin = GPIO_PIN_15;

```

```

00203     HAL_GPIO_Init(GPIOB, &gpio);
00204 }
00205
00206 void setColsOut(){
00207     //set columns to write
00208     GPIO_InitTypeDef gpio;
00209     gpio.Mode = GPIO_MODE_OUTPUT_PP;
00210     gpio.Pull = GPIO_PULLDOWN; //
00211     gpio.Speed = GPIO_SPEED_HIGH; //
00212     // D8
00213     gpio.Pin = GPIO_PIN_2;
00214     HAL_GPIO_Init(GPIOI, &gpio);
00215
00216     // D9
00217     gpio.Pin = GPIO_PIN_15;
00218     HAL_GPIO_Init(GPIOA, &gpio);
00219
00220     // D10
00221     gpio.Pin = GPIO_PIN_8;
00222     HAL_GPIO_Init(GPIOA, &gpio);
00223
00224     // D11
00225     gpio.Pin = GPIO_PIN_15;
00226     HAL_GPIO_Init(GPIOB, &gpio);
00227 }
00228
00229 int getInput (void){
00230     unsigned int k, r;
00231
00232     //set rows high one at a time
00233     for (r = 12; r <= 15; r++){
00234
00235         //set columns to low
00236         setColsOut();
00237         for (k = 8; k<=11; k++){
00238             turnOn(k);
00239         }
00240         setColsIn();
00241
00242         turnOff(r);
00243
00244         // check the value of each column
00245         for (k = 8; k <= 11; k++){
00246             if(readPin(k) == GPIO_PIN_SET){
00247                 turnOn(r);
00248                 return convertPinsToNum(k, r);
00249             }
00250         }
00251         turnOn(r);
00252     }
00253
00254     return -1;
00255 }

```

2.8 keypad.h

```

00001 void initializeMembranePins (void);
00002 int getInput (void);

```

2.9 led_main.c

```

00001 /*
00002  *****
00003  Author: Chaya Punnasri, 100386454
00004  Date of creation: 30 April 2024
00005  Name: led_main.c
00006  Description: Initialise and set/reset selected ports and pins for the use of an LED
00007  *****
00008  */
00009
00010 #include <stdio.h> //include the library for a standard input and output
00011 #include "stm32f7xx_hal.h" //include the library for HAL
00012 #include "stm32f7xx_hal_gpio.h" // include the HAL library for a GPIO use
00013 #include "led_main.h" // include the header file called led_main.h
00014
00015 // LED Red and Green gpio initialisation
00016 void initialise_led(void){
00017
00018     // this is for pin and port of green

```

```

00019     GPIO_InitTypeDef gpio; // initialise the type definition
00020     __HAL_RCC_GPIOI_CLK_ENABLE(); // enable clock for port I
00021     gpio.Pin = GPIO_PIN_0; // using pin 0
00022     gpio.Mode = GPIO_MODE_OUTPUT_PP; // output mode
00023     gpio.Pull = GPIO_NOPULL; // no pull for the GPIO output
00024     gpio.Speed = GPIO_SPEED_HIGH; // set the speed to high
00025     HAL_GPIO_Init(GPIOI, &gpio); // initialise port I pin 0
00026
00027     // this is for pin and port of red
00028     __HAL_RCC_GPIOG_CLK_ENABLE(); // enable the clock for port G
00029     gpio.Pin = GPIO_PIN_7; // using pin 7
00030     gpio.Mode = GPIO_MODE_OUTPUT_PP; //output mode
00031     gpio.Pull = GPIO_NOPULL; // no pull for the GPIO output
00032     gpio.Speed = GPIO_SPEED_HIGH; // set the speed to high
00033     HAL_GPIO_Init(GPIOG, &gpio); // initialise port G pin 7
00034 }
00035
00036 // function to set or clear the selected data port bit of a red colour
00037 void turnOnRed(void){
00038
00039     // set port G pin 7
00040     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_SET);
00041     // clear/reset port I pin 0
00042     HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, GPIO_PIN_RESET);
00043 }
00044
00045 // function to set or clear the selected data port bit of a green colour
00046 void turnOnGreen(void){
00047
00048     // set port I pin 0
00049     HAL_GPIO_WritePin(GPIOI, GPIO_PIN_0, GPIO_PIN_SET);
00050     // clear/reset port G pin 7
00051     HAL_GPIO_WritePin(GPIOG, GPIO_PIN_7, GPIO_PIN_RESET);
00052 }
00053
00054

```

2.10 led_main.h

```

00001 /*
00002  *****
00003  Author: Chaya Punnasri, 100386454
00004  Date of creation: 30 April 2024
00005  Name: led_main.h
00006  Description: It provides function prototypes to be used in other c file for a further implementation
00007  *****
00008  */
00009
00010 // providing a function prototype for the initialise_led() function
00011 void initialise_led(void);
00012
00013 // providing a function prototype for the turnOnRed() function
00014 void turnOnRed(void);
00015
00016 // providing a function prototype for thw turnOnGreen() function
00017 void turnOnGreen(void);

```

2.11 photoresistor.c

```

00001 /*
00002  *****
00003  Author: Chaya Punnasri, 100386454
00004  Date of creation: 5 May 2024
00005  Name: photoresistor.c
00006  Description: Initialise the system clock configuration for the RCC and the PWR which
00007  will be used in an analogue signal generation, after that the signal is converted into
00008  the digital signal
00009  *****
00010  */
00011
00012 #include <stdio.h>
00013 #include "stm32f7xx_hal.h"
00014 #include "GLCD_Config.h" // include the GLCD_config.h
00015 #include "Board_GLCD.h" // include the Board_GLCD.h
00016 #include "Board_Touch.h" // include Board_Touch.h
00017 #include "photoresistor.h" // include the photoresistor.h
00018 #define wait_delay HAL_Delay
00019 #include <math.h> // include the use of a math library
00020

```

```

00021 // calling the function from GLCD_Touch.h
00022 extern GLCD_FONT GLCD_Font_16x24;
00023
00024
00025 /*
00026 * System Clock Configuration
00027 */
00028
00029 void SystemClock_Config_Photoreistor(void) {
00030     RCC_OscInitTypeDef RCC_OscInitStruct;
00031     RCC_ClkInitTypeDef RCC_ClkInitStruct;
00032     /* Enable Power Control clock */
00033     __HAL_RCC_PWR_CLK_ENABLE();
00034     /* The voltage scaling allows optimizing the power
00035     consumption when the device is clocked below the
00036     maximum system frequency. */
00037     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
00038     /* Enable HSE Oscillator and activate PLL
00039     with HSE as source */
00040     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00041     RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00042     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00043     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00044     RCC_OscInitStruct.PLL.PLLM = 25;
00045     RCC_OscInitStruct.PLL.PLLN = 336;
00046     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00047     RCC_OscInitStruct.PLL.PLLQ = 7;
00048     HAL_RCC_OscConfig(&RCC_OscInitStruct);
00049     /* Select PLL as system clock source and configure
00050     the HCLK, PCLK1 and PCLK2 clocks dividers */
00051     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK |
00052     RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00053     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00054     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00055     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
00056     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
00057     HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
00058 }
00059
00060 // initialise the use of analogue to digital convertor
00061 ADC_HandleTypeDef hadc;
00062
00063 void MX_ADC_Init(void)
00064 {
00065     ADC_ChannelConfTypeDef sConfig;
00066
00067     /* Enable ADC CLOCK For every ADC */
00068     __HAL_RCC_ADC1_CLK_ENABLE();
00069     __HAL_RCC_ADC2_CLK_ENABLE();
00070     __HAL_RCC_ADC3_CLK_ENABLE();
00071
00072     /* Configure the global features of the ADC (Clock, Resolution, Data Alignment and number
00073     of conversion) */
00074     hadc.Instance = ADC3; // choosing ADC3 as we are using the port A pin 0
00075     hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
00076     hadc.Init.Resolution = ADC_RESOLUTION_10B;
00077     hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00078     hadc.Init.NbrOfConversion = 1;
00079     hadc.Init.ScanConvMode = ENABLE;
00080     hadc.Init.ContinuousConvMode = ENABLE;
00081     hadc.Init.DiscontinuousConvMode = DISABLE;
00082     HAL_ADC_Init(&hadc);
00083     //configure channel
00084     sConfig.Rank = 1;
00085     sConfig.Channel = ADC_CHANNEL_0; // choosing to use a channel 0 because it is compatible for port
    A pin 0
00086     sConfig.SamplingTime = ADC_SAMPLETIME_28CYCLES;
00087     HAL_ADC_ConfigChannel(&hadc, &sConfig);
00088     HAL_ADC_Start(&hadc);
00089     HAL_ADC_PollForConversion(&hadc, HAL_MAX_DELAY);
00090 }
00091
00092 // initialise a GPIO for analog
00093 GPIO_InitTypeDef GPIO_InitStruct;
00094 void MX_GPIO_Init(void){
00095     __HAL_RCC_GPIOA_CLK_ENABLE(); // enable clock for port A
00096     GPIO_InitStruct.Mode = GPIO_MODE_ANALOG; // configure to analog input mode
00097     GPIO_InitStruct.Pin = GPIO_PIN_0; // pin 0
00098     GPIO_InitStruct.Pull = GPIO_NOPULL;
00099     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
00100 }
00101
00102
00103
00104 // control LCD brightness
00105 void ctrl_brightness(float adc_read){
00106     float V_pwr=5.0; //configure the V_pwr

```

```

00107     float V_sense= (adc_read * V_pwr)/1024; //calculate the V_sense which 1024 is the reading range
00108     float brightness=V_sense/V_pwr; // calculate the ratio of brightness
00109
00110     if(brightness > 0.2)    {
00111         // toggle the dark mode
00112         GLCD_SetBackgroundColor(GLCD_COLOR_BLACK);
00113         GLCD_SetForegroundColor(GLCD_COLOR_WHITE);
00114         GLCD_ClearScreen();
00115     }
00116     else {
00117         // toggle the light mode
00118         GLCD_SetBackgroundColor(GLCD_COLOR_WHITE);
00119         GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00120         GLCD_ClearScreen();
00121     }
00122 }
00123 }
00124
00125 // main loop for a photoresistor
00126 int photoresistor_main(void){
00127
00128     uint16_t adc_read; // declare a variable for the ADC reading
00129     adc_read = HAL_ADC_GetValue(&hadc);
00130     SystemClock_Config_Photoresistor(); //Config Clocks
00131     MX_GPIO_Init();
00132     MX_ADC_Init();
00133     // read ADC value
00134     ctrl_brightness(adc_read);
00135     wait_delay(1);
00136 }

```

2.12 photoresistor.h

```

00001 /*
00002  ****
00003  Author: Chaya Punnasri, 100386454
00004  Date of creation: 5 May 2024
00005  Name: photoresistor.h
00006  Description: It provides function prototypes to be used in other c file for a further implementation
00007  ****
00008  */
00009
00010 int photoresistor_main(void);
00011 void MX_GPIO_Init(void);
00012 void MX_ADC_Init(void);
00013 void SystemClock_Config_Photoresistor(void);

```

2.13 Safe.c

```

00001 /*
00002  #####
00003  Principle Author:
00004
00005  Tommy Peach, 100358179
00006
00007  File name:
00008
00009  safe.c
00010
00011  Description:
00012
00013  safe.c is responsible for initialising all of the components that are not already initialised and
00014  also checks for touch screen presses
00015  #####
00016  */
00017 #include "GLCDTOUCH.h"
00018 #include "buzzer.h"
00019 #include "keypad.h"
00020 #include "photoresistor.h"
00021 #include "Board_Touch.h"
00022 #include "shock.h"
00023 #include "screens.h"
00024
00025 int main(void){
00026
00027     initializeMembranePins();
00028     HAL_Init(); //Init Hardware Abstraction Layer
00029

```

```

00030         SystemClock_Config();
00031         Touch_Initialize();
00032         GLCD_Initialize();
00033         TIM2_Init();
00034         PA15_Init();
00035         initialise_input();
00036         initialise_led();
00037         MX_GPIO_Init();
00038         MX_ADC_Init();
00039         SystemClock_Config_Photoresistor();
00040         __HAL_RCC_GPIOC_CLK_ENABLE();
00041         __USART6_CLK_ENABLE();
00042         init_shock();
00043         init_buzzer();
00044         changeScreen(0); // go to set pin screen
00045
00046         for(;;){
00047             Touch_GetState(&tsc_state);
00048             if(tsc_state.pressed){
00049                 tsc_state.pressed = 0;
00050                 CheckCoords(tsc_state.x, tsc_state.y);
00051                 continue;
00052             }
00053         }
00054     }

```

2.14 screens.c

```

00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     screens.c
00010
00011     Description:
00012
00013     Screens.c handles the screens, specifically drawing the elements on each screen, as well as
00014     handling the transition between screens through CheckCoords.
00015     From the stm website, I was able to determine that the resolution of the screen was 480x272
00016     pixels, so this is the number the calculations were based on for centering the text.
00017
00018     Calculations for centering text goes as follows:
00019
00020     (16 pixels (size of the font width) * (number of letters)) / 2) - 240.
00021     This calculation takes the midpoint of the sentence/text and minuses half of the screen size to
00022     find the point where the text should start.
00023     #####
00024 */
00025 #include "GLCD_Config.h"
00026 #include "Board_GLCD.h"
00027 #include "GLCDTOUCH.h"
00028 int ScreenState = 0; // Initialising ScreenState, 0 represents the first screen which is the set pin
00029 screen
00030
00031 /* CheckCoords checks the coordinates that a user has pressed on the screen, and passes these values
00032 into if statements.
00033 * The conditions inside of the if statements make sure that the correct coordinates are pressed for
00034 the back buttons on each screen.
00035 * It also checks for the "Unlock" and "Reset Pin" buttons on the home screen.
00036 * Each if statement includes the ScreenState for which screens these conditions should apply for.
00037 This avoids changing the screen for unintended screens.
00038 */
00039
00040 void CheckCoords(int x, int y){
00041     // If statement to transition back to the home screen
00042     if ((x > 0 && x < 70) && (y > 0 && y < 25) && (ScreenState == 2 || ScreenState == 3
00043         || ScreenState == 4 || ScreenState==5 )){
00044         changeScreen(1);
00045     }
00046     // If statement for transition to the unlock screen
00047     if ((x < 288 && x > 192) && (y < 140 && y > 116) && ScreenState == 1){
00048         changeScreen(2);
00049     }
00050     // If statement for transition to the reset pin screen
00051     if ((x < 312 && x > 168) && (y < 92 && y > 68) && ScreenState == 1){
00052         changeScreen(3);
00053     }
00054     // Transition for the set pin confirmation screen, if the user wants to change the pin on the
00055     first time setting

```

```

00049     if ((x > 0 && x < 70) && (y > 0 && y < 25) && (ScreenState == 6)){
00050         changeScreen(0);
00051     }
00052 }
00053
00054
00055 void unlockScreen(){
00056     int x = 37;
00057     int i = 0;
00058     GLCD_DrawString(192, 0*24, "UNLOCK");
00059     GLCD_DrawString(104 , 3*24, "Input Combination");
00060     GLCD_DrawString(0*24, 0*24, "Back");
00061     GLCD_DrawRectangle(0*24, 0*24, 65, 25); // Drawing back button box
00062     // Drawing input boxes
00063     for (i = 0; i <= 7; i++) {
00064         x += 42;
00065         GLCD_DrawString(x, 136, "_");
00066     }
00067 }
00068
00069 void resetScreen(){
00070     int x = 37;
00071     int i = 0;
00072     GLCD_DrawString(168, 0*24, "Reset Pin");
00073     GLCD_DrawString(40 , 3*24, "Input current combination");
00074     GLCD_DrawString(0*24, 0*24, "Back");
00075     GLCD_DrawRectangle(0*24, 0*24, 65, 25); // Drawing back button box
00076
00077     // Drawing input boxes
00078     for (i = 0; i <= 7; i++){
00079         x += 42;
00080         GLCD_DrawString(x, 136, "_");
00081     }
00082 }
00083
00084 void hscreen(){
00085     GLCD_DrawString(192, 116, "Unlock");
00086     GLCD_DrawString(168, 68, "Reset Pin");
00087 }
00088
00089
00090 void lockedScreen(){
00091     GLCD_DrawString(72, 4*24, "Please close door to");
00092     GLCD_DrawString(40, 5*24, "automatically lock system");
00093 }
00094
00095 void confirmScreen(){
00096     int x = 37;
00097     int i = 0;
00098     GLCD_DrawString(152, 0*24, "Confirm Pin");
00099     GLCD_DrawString(88, 3*24, "Confirm combination");
00100     GLCD_DrawString(0*24, 0*24, "Back");
00101     GLCD_DrawRectangle(0*24, 0*24, 65, 25);
00102     // Drawing input boxes
00103     for (i = 0; i <= 7; i++){
00104         x += 42;
00105         GLCD_DrawString(x, 136, "_");
00106     }
00107 }
00108 void resetSetScreen(){
00109     int x = 37;
00110     int i = 0;
00111     GLCD_DrawString(168, 0*24, "Reset Pin");
00112     GLCD_DrawString(72, 3*24, "Input new combination");
00113     GLCD_DrawString(0*24, 0*24, "Back");
00114     GLCD_DrawRectangle(0*24, 0*24, 65, 25);
00115
00116     // Drawing input boxes
00117     for (i = 0; i <= 7; i++){
00118         x += 42;
00119         GLCD_DrawString(x, 136, "_");
00120     }
00121 }
00122 void setPinScreen(){
00123     int x = 37;
00124     int i = 0;
00125     GLCD_DrawString(184, 0*24, "Set pin");
00126     GLCD_DrawString(104, 3*24, "Input combination");
00127     // Drawing input boxes
00128     for (i = 0; i <= 7; i++){
00129         x += 42;
00130         GLCD_DrawString(x, 136, "_");
00131     }
00132     ScreenState = 0;
00133 }
00134 void setPinConfirmScreen(){
00135     int x = 37;

```

```

00136     int i = 0;
00137     GLCD_DrawString(184, 0*24, "Set pin");
00138     GLCD_DrawString(104, 3*24, "Confirm combination");
00139     GLCD_DrawString(0*24, 0*24, "Back");
00140     GLCD_DrawRectangle(0*24, 0*24, 65, 25);
00141     // Drawing input boxes
00142     for (i = 0; i <= 7; i++){
00143         x += 42;
00144         GLCD_DrawString(x, 136, "_");
00145     }
00146 }

```

2.15 screens.h

```

00001
00002 /*
00003     #####
00004     Principle Author:
00005
00006     Tommy Peach, 100358179
00007
00008     File name:
00009
00010     screens.h
00011
00012     Description:
00013
00014     screens.h defines functions to be used in other source files.
00015     #####
00016 */
00017 #include "GLCD_Config.h"
00018 #include "Board_GLCD.h"
00019 #include "String.h"
00020 extern int ScreenState;
00021
00022 void unlockScreen();
00023 void resetScreen();
00024 void homeScreen();
00025 void hscreen();
00026 void lockedScreen();
00027 void confirmScreen();
00028 void resetSetScreen();
00029 void setPinScreen();
00030 void setPinConfirmScreen();
00031 int CheckCoords(int x, int y);

```

2.16 servo.c

```

00001 /*
00002     *****
00003     Author: Chaya Punnasri, 100386454
00004     Date of creation: 25 April 2024
00005     Name: servo.c
00006     Description: The servo will be working by initialise the GPIO pin PB4 which supports
00007     the PWM and that pin is using TIM3 channel 1. The servo turns different angles
00008     *****
00009 */
00010
00011 #include "stm32f7xx_hal.h" // HAL library is being used
00012 #include "servo.h"
00013
00014 TIM_HandleTypeDef htim2;
00015
00016 void PA15_Init(){
00017     GPIO_InitTypeDef gpio;
00018
00019     __HAL_RCC_GPIOB_CLK_ENABLE(); // enable the clock for a base B
00020
00021     gpio.Pin = GPIO_PIN_4; // pin 4
00022     gpio.Mode = GPIO_MODE_AF_PP;
00023     gpio.Pull = GPIO_NOPULL;
00024     gpio.Speed = GPIO_SPEED_HIGH;
00025     gpio.Alternate = GPIO_AF2_TIM3;
00026     HAL_GPIO_Init(GPIOB, &gpio);
00027 }
00028
00029
00030 void TIM2_Init(void){
00031     TIM_ClockConfigTypeDef sClockSourceConfig;

```



```

00035     TIM_OC_InitTypeDef sConfigOC;
00036
00037     //Timer configuration
00038     htim2.Instance = TIM3;
00039     htim2.Init.Prescaler = 32000; // set prescaler to 32000 and the signal 500Hz on the TIM3
00040     htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
00041     htim2.Init.Period = 9; // set period to 9 then it becomes 10, so the frequency will be 50 Hz cycle
00042     htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00043
00044     HAL_TIM_Base_Init(&htim2);
00045
00046     //Set the timer in PWM mode
00047     sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
00048     HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig);
00049     HAL_TIM_PWM_Init(&htim2);
00050
00051     //Channel configuration
00052     sConfigOC.OCMode = TIM_OCMODE_PWM1;
00053     sConfigOC.Pulse = 0;
00054     sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
00055     sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00056     HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1);
00057
00058     PA15_Init();
00059 }
00060
00061 // opened angle which is 90 degrees
00062 void opened_angle(void) {
00063
00064     //Reset of all peripherals, initializes the Flash interface and the SysTick
00065     HAL_Init();
00066
00067     //Enable peripheral clock for TIM3
00068     __HAL_RCC_TIM3_CLK_ENABLE();
00069
00070     //Initialize TIM3, CH1 and PB4
00071     TIM2_Init();
00072
00073     //Start PWM on TIM3_CH1
00074     HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
00075
00076     // change the CCR to 1 so the output will be high
00077     htim2.Instance->CCR1 = 1;
00078
00079     // perform a delay for 1 second
00080     HAL_Delay(1000);
00081 }
00082
00083 // closed angle which is 0 degree
00084 int closed_angle(void) {
00085
00086     //Reset of all peripherals, initializes the Flash interface and the SysTick
00087     HAL_Init();
00088
00089     //Enable peripheral clock for TIM3
00090     __HAL_RCC_TIM3_CLK_ENABLE();
00091
00092     //Initialize TIM3, CH1 and PB4
00093     TIM2_Init();
00094
00095     //Start PWM on TIM3_CH1
00096     HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
00097
00098     // change the CCR to 3 so the output will be low
00099     htim2.Instance->CCR1 = 3;
00100
00101     //perform a delay for a second
00102     HAL_Delay(1000);
00103 }

```

2.17 servo.h

```

00001 /*
00002  *****
00003  Author: Chaya Punnasri, 100386454
00004  Date of creation: 25 April 2024
00005  Name: servo.h
00006  Description: It provides function prototypes to be used in other c file for a further implementation
00007  *****
00008  */
00009
00010 void PA15_Init();
00011 void TIM2_Init(void);

```

```

00012 void opened_angle(void);
00013 void closed_angle(void);
00014
00015 // link to the function in GLCD_touch.c
00016 void SystemClock_Config(void);

```

2.18 shock.c

```

00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     shock.c
00010
00011     Description:
00012
00013     shock.c is responsible for initialising the shock sensor. It also sets the priority for the
00014     interrupt for the shock sensor and enables the handler for the shock sensor.
00015     #####
00016 */
00017 #include <stdio.h>
00018 #include "stm32f7xx_hal.h"
00019 #include "stm32f7xx_hal_gpio.h"
00020 #include "shock.h"
00021 void init_shock(void) {
00022
00023     GPIO_InitTypeDef gpio;
00024     __HAL_RCC_GPIOG_CLK_ENABLE();
00025     gpio.Pin = GPIO_PIN_6; // Setting relevant pin 6 for GPIOG
00026
00027     /* Interrupt mode, detects interrupt if the voltage goes low to high.
00028     * This seems to make sense as after looking up how the shock sensor works, it seems once the
00029     inside coil touches the outer casing it will return high.
00030     * This mode was found in stm32f7xx_hal_gpio.h.
00031     */
00032     gpio.Mode = GPIO_MODE_IT_RISING;
00033
00034     gpio.Pull = GPIO_NOPULL;
00035     gpio.Speed = GPIO_SPEED_HIGH;
00036     HAL_GPIO_Init(GPIOG, &gpio);
00037
00038     /* Setting the interrupt priority, 0 being the highest priority as stated in lecture notes.
00039     * According to the reference manual
00040     (https://learn.uea.ac.uk/bbcswebdav/pid-4513906-dt-content-rid-29049770\_1/xid-29049770\_1, page 296)
00041     * and the vector table in startup_stm32f746xx.s,
00042     * EXTI9_5_IRQn seems to be the interrupt service handler for pins 5 - 9, as shock is pin 6.
00043     */
00044     HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
00045     // Enabling the interrupt service
00046     HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
00047 }

```

2.19 shock.h

```

00001 /*
00002     #####
00003     Principle Author:
00004
00005     Tommy Peach, 100358179
00006
00007     File name:
00008
00009     shock.h
00010
00011     Description:
00012
00013     shock.h declares functions to be used in other source files.
00014     #####
00015 */
00016
00017 void init_shock(void);

```

2.20 tracking.c

```

00001 /*
00002 *****
00003 Author: Chaya Punnasri, 100386454
00004 Date of creation: 8 May 2024
00005 Name: tracking.c
00006 Description: The tracking sensor is being initialised and being used as an input GPIO
00007 *****
00008 */
00009
00010 #include <stdio.h>
00011 #include "stm32f7xx_hal.h"
00012 #include "stm32f7xx_hal_gpio.h"
00013
00014 void initialise_input(void){
00015     // tracking sensor port initialisation
00016     GPIO_InitTypeDef gpio;
00017     __HAL_RCC_GPIOI_CLK_ENABLE(); // enable clock for base I
00018     gpio.Pin = GPIO_PIN_3; // pin 3
00019     gpio.Mode = GPIO_MODE_INPUT; // set mode to input as it needs to detect for the object moving
00020     nearby
00021     gpio.Pull = GPIO_PULLUP;
00022     gpio.Speed = GPIO_SPEED_HIGH;
00023     HAL_GPIO_Init(GPIOI, &gpio);
00024 }

```

2.21 tracking.h

```

00001 /*
00002 *****
00003 Author: Chaya Punnasri, 100386454
00004 Date of creation: 8 May 2024
00005 Name: tracking.h
00006 Description: It provides function prototypes to be used in other c file for a further implementation
00007 *****
00008 */
00009
00010 void initialise_input(void);

```


Index

EmbeddedSummative/BUZZER.c, [3](#)
EmbeddedSummative/buzzer.h, [3](#)
EmbeddedSummative/GLCDTOUCH.c, [4](#)
EmbeddedSummative/GLCDTOUCH.h, [9](#)
EmbeddedSummative/interrupt.c, [10](#)
EmbeddedSummative/interrupt.h, [11](#)
EmbeddedSummative/keypad.c, [11](#)
EmbeddedSummative/keypad.h, [14](#)
EmbeddedSummative/led_main.c, [14](#)
EmbeddedSummative/led_main.h, [15](#)
EmbeddedSummative/photoresistor.c, [15](#)
EmbeddedSummative/photoresistor.h, [17](#)
EmbeddedSummative/Safe.c, [17](#)
EmbeddedSummative/screens.c, [18](#)
EmbeddedSummative/screens.h, [20](#)
EmbeddedSummative/servo.c, [20](#)
EmbeddedSummative/servo.h, [21](#)
EmbeddedSummative/shock.c, [22](#)
EmbeddedSummative/shock.h, [22](#)
EmbeddedSummative/tracking.c, [23](#)
EmbeddedSummative/tracking.h, [23](#)