```c
/*
 * verifySequence.c
 *
 *  Created on: Jun 4, 2015
 *      Author: Taylor Cowley
 */

#include "verifySequence.h"

//The flag that shows whether we are enabled
bool verifySequence_enable_flag = false;

//The flag that shows whether we have completed the sequence
bool verifySequence_completed_flag = false;

//The flag that shows whether the user timed out
bool verifySequence_time_out = false;

//The flag that shows whether the user failed the sequence
bool verifySequence_user_fail = false;

//this stores the current state of our MACHINE
verifySequence_st_t verifySequence_currentState = verifySequence_init_st;



// State machine will run when enabled.
void verifySequence_enable(){
    verifySequence_enable_flag = true;
}

// This is part of the interlock. You disable the state-machine and then enable it again.
void verifySequence_disable(){
    verifySequence_enable_flag = false;
    verifySequence_currentState = verifySequence_init_st;
}

// Used to detect if there has been a time-out error.
bool verifySequence_isTimeOutError(){
    return verifySequence_time_out;
}

// Used to detect if the user tapped the incorrect sequence.
bool verifySequence_isUserInputError(){
    return verifySequence_user_fail;
}

// Used to detect if the verifySequence state machine has finished verifying.
bool verifySequence_isComplete(){
    return verifySequence_completed_flag;
}

// Standard tick function.
void verifySequence_tick(){

    //this stores the wait timer for wait_for_touch and touch_cooldown
    static int16_t delay_timer = 0;
```

```
58
59      //this is the index where we currently are flashing
60      static int16_t current_index = 0;
61
62
63      //first we do state functions
64      switch(verifySequence_currentState){
65      case verifySequence_init_st://Init everything
66          current_index = 0;                      //start at the beginning; a very good place to
    start
67          verifySequence_completed_flag = false;  //We haven't completed
68          verifySequence_time_out = false;        //we haven't timed out
69          verifySequence_user_fail = false;       //we haven't failed
70          break;
71
72      case wait_for_enable_v:     //we can't do anything unless enabled
73          //so do nothing
74          break;
75
76      case wait_for_touch:        //wait for the user to touch a button
77          delay_timer--;              //countdown the timeout timer
78          break;
79
80      case wait_for_release:      //we wait button handler to register a release
81          break;
82
83      case analyze_touch:         //time to register the touch
84          buttonHandler_disable();    //we need to disable/reset the buttonHandler
85          if(buttonHandler_getRegionNumber() != globals_getSequenceValue(current_index)){
86              //they pushed the wrong button! :(
87              verifySequence_user_fail = true;
88          }
89          break;
90
91      case end_verify_sequence:   //we have ended the verify sequence
92          verifySequence_completed_flag = true;   //we flag that we have finished
93          break;
94
95      case wait_for_disable_v:    //chill here until disabled
96          //so we do nothing
97          break;
98
99      default:                    //error in state
100         printf("We have reached an impossible state");
101         break;
102     }
103
104
105
106     //then we do state transitions
107     switch(verifySequence_currentState){
108     case verifySequence_init_st://Init everything (like the screen)
109         verifySequence_currentState = wait_for_enable_v;        //We inited everything! next
    state
110         break;
111
112     case wait_for_enable_v:     //we can't do anything unless enabled
```

```
113         if(verifySequence_enable_flag){                     //are we enabled?
114             verifySequence_currentState = wait_for_touch;   //yes! move on
115             delay_timer = VERIFYSEQUENCE_TIMEOUT_SPEED;     //start the timer for user timeout
116             buttonHandler_enable();                         //let's enable the button handler
    now!
117         }
118         break;
119
120     case wait_for_touch:        //wait for the user to touch a button
121         if(display_isTouched()){                            //They touched in time!
122             verifySequence_currentState = wait_for_release; //start the touch sensor cooldown
123         }
124         if(delay_timer <= 0){               //aww user timeout :(
125             verifySequence_currentState = end_verify_sequence;      //I guess we go to the end
126             verifySequence_time_out = true;                         //record that the user
    timed out
127             buttonHandler_disable();                                //We should disable the
    buttons
128         }
129         break;
130
131     case wait_for_release:      //we wait for the user to release (button handler will tell
    us)
132         if(buttonHandler_releaseDetected()){                    //the user let go
133             verifySequence_currentState = analyze_touch;        //now let's analyze her touch
134             buttonHandler_disable();                            //we should disable the
    buttons
135         }
136         break;
137
138     case analyze_touch:         //time to register the touch
139         //if we have are done
140         if(verifySequence_isUserInputError()                                    //if the user
    failed
141                 || verifySequence_isTimeOutError()                              //or the user
    timed out
142                 || current_index >= globals_getSequenceIterationLength() - 1){  //or we are at
    the end of the sequence
143             verifySequence_currentState = end_verify_sequence;      //move on to end!
144         }else{          //the sequence is NOT over
145             verifySequence_currentState = wait_for_touch;           //wait for the next touch
146             current_index++;                        //move on to next item in sequence
147             buttonHandler_enable();                 //turn the button handler back on
148         }
149         break;
150
151     case end_verify_sequence:   //we have ended the verify sequence
152         verifySequence_currentState = wait_for_disable_v;           //only one tick in this
    state
153         break;
154
155     case wait_for_disable_v:        //chill here until disabled
156         if(!verifySequence_enable_flag){                            //we are disabled
157             verifySequence_currentState = verifySequence_init_st;   //go back to the
    beginning!
158         }
159         break;
```

```
160
161     default:                    //This is an error; print it
162         printf("impossible state found");
163         break;
164     }
165
166
167 }
168
169
170
171
172 #define MESSAGE_X 0
173 #define MESSAGE_Y (display_width()/4)
174 #define MESSAGE_TEXT_SIZE 2
175 #define MESSAGE_STARTING_OVER
176 // Prints the instructions that the user should follow when
177 // testing the verifySequence state machine.
178 // Takes an argument that specifies the length of the sequence so that
179 // the instructions are tailored for the length of the sequence.
180 // This assumes a simple incrementing pattern so that it is simple to
181 // instruct the user.
182 void verifySequence_printInstructions(uint8_t length, bool startingOver) {
183   display_fillScreen(DISPLAY_BLACK);        // Clear the screen.
184   display_setTextSize(MESSAGE_TEXT_SIZE);   // Make it readable.
185   display_setCursor(MESSAGE_X, MESSAGE_Y);  // Rough center.
186   if (startingOver) {                       // Print a message if you start over.
187     display_fillScreen(DISPLAY_BLACK);      // Clear the screen if starting over.
188     display_setTextColor(DISPLAY_WHITE);    // Print whit text.
189     display_println("Starting Over. ");
190   }
191   display_println("Tap: ");
192   display_println();
193   switch (length) {
194   case 1:
195     display_println("red");
196     break;
197   case 2:
198     display_println("red, yellow ");
199     break;
200   case 3:
201     display_println("red, yellow, blue ");
202     break;
203   case 4:
204     display_println("red, yellow, blue, green ");
205     break;
206   default:
207     break;
208   }
209   display_println("in that order.");
210   display_println();
211   display_println("hold BTN0 to quit.");
212 }
213
214 // Just clears the screen and draws the four buttons used in Simon.
215 void verifySequence_drawButtons() {
216     display_fillScreen(DISPLAY_BLACK);
```

```
217     simonDisplay_drawAllButtons();  // Draw the four buttons.
218 }
219
220 // This will set the sequence to a simple sequential pattern.
221 #define MAX_TEST_SEQUENCE_LENGTH 4  // the maximum length of the pattern
222 uint8_t verifySequence_testSequence[MAX_TEST_SEQUENCE_LENGTH] = {0, 1, 2, 3};  // A simple
    pattern.
223 #define MESSAGE_WAIT_MS 4000  // Display messages for this long.
224
225 // Increment the sequence length making sure to skip over 0.
226 // Used to change the sequence length during the test.
227 int16_t incrementSequenceLength(int16_t sequenceLength) {
228   int16_t value = (sequenceLength + 1) % (MAX_TEST_SEQUENCE_LENGTH+1);
229   if (value == 0) value++;
230     return value;
231 }
232
233 // Used to select from a variety of informational messages.
234 enum verifySequence_infoMessage_t {
235   user_time_out_e,            // means that the user waited too long to tap a color.
236   user_wrong_sequence_e,      // means that the user tapped the wrong color.
237   user_correct_sequence_e,    // means that the user tapped the correct sequence.
238   user_quit_e                 // means that the user wants to quite.
239 };
240
241 // Prints out informational messages based upon a message type (see above).
242 void verifySequence_printInfoMessage(verifySequence_infoMessage_t messageType) {
243   // Setup text color, position and clear the screen.
244   display_setTextColor(DISPLAY_WHITE);
245   display_setCursor(MESSAGE_X, MESSAGE_Y);
246   display_fillScreen(DISPLAY_BLACK);
247   switch(messageType) {
248   case user_time_out_e:  // Tell the user that they typed too slowly.
249     display_println("Error:");
250     display_println();
251     display_println("  User tapped sequence");
252     display_println("  too slowly.");
253     break;
254   case user_wrong_sequence_e:  // Tell the user that they tapped the wrong color.
255     display_println("Error: ");
256     display_println();
257     display_println("  User tapped the");
258     display_println("  wrong sequence.");
259     break;
260   case user_correct_sequence_e:  // Tell the user that they were correct.
261     display_println("User tapped");
262     display_println("the correct sequence.");
263     break;
264   case user_quit_e:             // Acknowledge that you are quitting the test.
265     display_println("quitting runTest().");
266     break;
267   default:
268     break;
269   }
270 }
271
272 #define BTN0 1
```

```
273 // Tests the verifySequence state machine.
274 // It prints instructions to the touch-screen. The user responds by tapping the
275 // correct colors to match the sequence.
276 // Users can test the error conditions by waiting too long to tap a color or
277 // by tapping an incorrect color.
278 void verifySequence_runTest() {
279   display_init();  // Always must do this.
280   buttons_init();  // Need to use the push-button package so user can quit.
281   int16_t sequenceLength = 1;  // Start out with a sequence length of 1.
282   verifySequence_printInstructions(sequenceLength, false);  // Tell the user what to do.
283   utils_msDelay(MESSAGE_WAIT_MS);  // Give them a few seconds to read the instructions.
284   verifySequence_drawButtons();    // Now, draw the buttons.
285   // Set the test sequence and it's length.
286   globals_setSequence(verifySequence_testSequence, MAX_TEST_SEQUENCE_LENGTH);
287   globals_setSequenceIterationLength(sequenceLength);
288   // Enable the verifySequence state machine.
289   verifySequence_enable();           // Everything is interlocked, so first enable the
    machine.
290   while (!(buttons_read() & BTN0)) { // Need to hold button until it quits as you might be
    stuck in a delay.
291     // verifySequence uses the buttonHandler state machine so you need to "tick" both of them.
292     verifySequence_tick();  // Advance the verifySequence state machine.
293     buttonHandler_tick();   // Advance the buttonHandler state machine.
294     utils_msDelay(1);       // Wait 1 ms.
295     // If the verifySequence state machine has finished, check the result, otherwise just keep
    ticking both machines.
296     if (verifySequence_isComplete()) {
297       if (verifySequence_isTimeOutError()) {                      // Was the user too slow?
298         verifySequence_printInfoMessage(user_time_out_e);         // Yes, tell the user that
    they were too slow.
299       } else if (verifySequence_isUserInputError()) {             // Did the user tap the
    wrong color?
300         verifySequence_printInfoMessage(user_wrong_sequence_e);   // Yes, tell them so.
301       } else {
302         verifySequence_printInfoMessage(user_correct_sequence_e); // User was correct if you
    get here.
303       }
304       utils_msDelay(MESSAGE_WAIT_MS);                           // Allow the user to read the
    message.
305       sequenceLength = incrementSequenceLength(sequenceLength);  // Increment the sequence.
306       globals_setSequenceIterationLength(sequenceLength);  // Set the length for the
    verifySequence state machine.
307       verifySequence_printInstructions(sequenceLength, true);    // Print the instructions.
308       utils_msDelay(MESSAGE_WAIT_MS);                           // Let the user read the
    instructions.
309       verifySequence_drawButtons();                             // Draw the buttons.
310       verifySequence_disable();                                 // Interlock: first step of
    handshake.
311       verifySequence_tick();                                    // Advance the verifySequence
    machine.
312       utils_msDelay(1);                                         // Wait for 1 ms.
313       verifySequence_enable();                                  // Interlock: second step of
    handshake.
314       utils_msDelay(1);                                         // Wait 1 ms.
315     }
316   }
317   verifySequence_printInfoMessage(user_quit_e);  // Quitting, print out an informational
```

```
   message.
318 }
319
```