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

#include "flashSequence.h"


//The flag that shows whether we are enabled or not
bool flashSequence_enable_flag = false;

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

// Turns on the state machine. Part of the interlock.
void flashSequence_enable(){
    flashSequence_enable_flag = true;   //enable = true the flag!
}

// Turns off the state machine. Part of the interlock.
void flashSequence_disable(){
    flashSequence_enable_flag = false;  //disable = false the flag!
}

// Other state machines can call this to determine if this state machine is finished.
bool flashSequence_completed(){
    return flashSequence_completed_flag;    //so we return the flag.
}

// Standard tick function.
void flashSequence_tick(){
    //Stores the current state of the state machine
    static flashSequence_st_t currentState = flashSequence_init_st;

    //This is the timer to delay for displaying/blanking the squares
    static int16_t delay_timer = 0;

    //this is the length of the sequence we are currently flashing
    static int16_t sequence_length = 0;

    //this is the index where we currently are flashing
    static int16_t current_index = 0;

    //first we do the state actions
    switch(currentState){
    case flashSequence_init_st:                 //Init anything
        current_index = 0;                      //we are displaying the sequence starting at 0
        //we need to know the current sequence length.

        flashSequence_completed_flag = false;   //we have not completed a sequence
        break;

    case wait_for_enable:           //we can't do anything unless enabled
        //The iteration length might be changed right before we are enabled, so check it here.
        sequence_length = globals_getSequenceIterationLength();
```

```
58          break;
59
60      case display_current_square:    //flash the current square of the sequence
61          delay_timer--;      //we chill here until the delay timer is gone
62          break;
63
64      case blank:                     //a blank in between every flash
65          delay_timer--;      //we also chill here a while
66          break;
67
68      case end_flash_sequence:        //we have ended the flash sequence
69          flashSequence_completed_flag = true;    //record that we have completed it
70          break;
71
72      case wait_for_disable:          //chill here until disabled
73          //which means we do nothing.
74          break;
75
76      default:                        //this is an error
77          printf("Invalid state!");
78          break;
79      }
80
81      //and lastly we do the state change
82      switch(currentState){
83      case flashSequence_init_st:                 //Init anything (like the screen)
84          currentState = wait_for_enable;     //chill in init only one tick
85          break;
86
87      case wait_for_enable:           //we can't do anything unless enabled
88          if(flashSequence_enable_flag){      //we are enabled; move to next state
89              delay_timer = FLASHSEQUENCE_BLINK_SPEED;    //start the countdown for the blink
90              currentState = display_current_square;      //and move to displaying
91              //we also display the current square on the screen during this time.
92              simonDisplay_drawSquare(globals_getSequenceValue(current_index), false);
93          }
94          break;
95
96      case display_current_square:    //flash the current square of the sequence
97          if(delay_timer <= 0){               //Have we reached our countdown?
98              delay_timer = FLASHSEQUENCE_BLANK_SPEED;    //yes! time to do blank countdown!
99              currentState = blank;           //move to the blank in between blinks
100
101             //we also need to draw a blank where we were drawing a square before
102             simonDisplay_drawSquare(globals_getSequenceValue(current_index), true);
103         }
104         break;
105
106     case blank:                     //a blank in between every flash
107         if(delay_timer <= 0){               //have we reached our countdown?
108             if(current_index < sequence_length -1){ //have we finished the sequence yet?
109                 current_index++;                    //we have not! increment the index
110
111                 //display the next square
112                 simonDisplay_drawSquare(globals_getSequenceValue(current_index), false);
113                 delay_timer = FLASHSEQUENCE_BLINK_SPEED;//and set the countdown for the next
    square
```

```
114                 currentState = display_current_square;  //and display the next square
115             } else {                                    //We have reached the end of the sequence
116                 currentState = end_flash_sequence;  //We have finished the sequence
117             }
118         }
119         break;
120 
121     case end_flash_sequence:        //we have ended the flash sequence
122         currentState = wait_for_disable;    //we do our things, then wait for disable
123         break;
124 
125     case wait_for_disable:          //chill here until disabled
126         if(!flashSequence_enable_flag){
127             currentState = flashSequence_init_st;           //we are disabled; move to next
    state
128         }
129         break;
130 
131     default:                        //this is an error
132         printf("Invalid state!");
133         break;
134     }
135 
136 }
137 
138 
139 
140 
141 // This will set the sequence to a simple sequential pattern.
142 // It starts by flashing the first color, and then increments the index and flashes the first
143 // two colors and so forth. Along the way it prints info messages to the LCD screen.
144 #define TEST_SEQUENCE_LENGTH 8  // Just use a short test sequence.
145 uint8_t flashSequence_testSequence[TEST_SEQUENCE_LENGTH] = {SIMON_DISPLAY_REGION_0,
146                                 SIMON_DISPLAY_REGION_1,
147                                 SIMON_DISPLAY_REGION_2,
148                                 SIMON_DISPLAY_REGION_3,
149                                 SIMON_DISPLAY_REGION_3,
150                                 SIMON_DISPLAY_REGION_2,
151                                 SIMON_DISPLAY_REGION_1,
152                                 SIMON_DISPLAY_REGION_0};
153 #define INCREMENTING_SEQUENCE_MESSAGE1 "Incrementing Sequence"  // Info message.
154 #define RUN_TEST_COMPLETE_MESSAGE "Runtest() Complete"      // Info message.
155 #define MESSAGE_TEXT_SIZE 2                                 // Make the text easy to see.
156 
157 // Print the incrementing sequence message.
158 void flashSequence_printIncrementingMessage() {
159   display_fillScreen(DISPLAY_BLACK);// Otherwise, tell the user that you are incrementing the
    sequence.
160   display_setCursor(0, display_height()/2);     // Roughly centered.
161   display_println(INCREMENTING_SEQUENCE_MESSAGE1);  // Print the message.
162   utils_msDelay(2000);                              // Hold on for 2 seconds.
163   display_fillScreen(DISPLAY_BLACK);            // Clear the screen.
164 }
165 
166 void flashSequence_runTest() {
167   display_init();   // We are using the display.
168   display_fillScreen(DISPLAY_BLACK);    // Clear the display.
```

```
169   globals_setSequence(flashSequence_testSequence, TEST_SEQUENCE_LENGTH);    // Set the
    sequence.
170   flashSequence_enable();                  // Enable the flashSequence state machine.
171   int16_t sequenceLength = 1;                           // Start out with a sequence of length
    1.
172   globals_setSequenceIterationLength(sequenceLength);   // Set the iteration length.
173   display_setTextSize(MESSAGE_TEXT_SIZE);           // Use a standard text size.
174   while (1) {                    // Run forever unless you break.
175     flashSequence_tick();   // tick the state machine.
176     utils_msDelay(1);   // Provide a 1 ms delay.
177     if (flashSequence_completed()) {  // When you are done flashing the sequence.
178       flashSequence_disable();  // Interlock by first disabling the state machine.
179       flashSequence_tick(); // tick is necessary to advance the state.
180       utils_msDelay(1);     // don't really need this here, just for completeness.
181       flashSequence_enable();   // Finish the interlock by enabling the state machine.
182       utils_msDelay(1); // Wait 1 ms for no good reason.
183       sequenceLength++; // Increment the length of the sequence.
184       if (sequenceLength > TEST_SEQUENCE_LENGTH) // Stop if you have done the full sequence.
185         break;
186       flashSequence_printIncrementingMessage();  // Tell the user that you are going to the
    next step in the pattern.
187       globals_setSequenceIterationLength(sequenceLength);   // Set the length of the pattern.
188     }
189   }
190   // Let the user know that you are finished.
191   display_fillScreen(DISPLAY_BLACK);
192   display_setCursor(0, display_height()/2);
193   display_println(RUN_TEST_COMPLETE_MESSAGE);
194 }
195
```