//simonMain.c

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "globals.h"

#include "simonDisplay.h"

#include "simonButtonHandler.h"

#include "simonFlashSequence.h"

#include "simonVerifySequence.h"

#include "SimonControl.h"

#include "buttons.h"

#define LARGE\_PRIME\_NUMBER 7919

#define TOTAL\_SECONDS 60

#define TIMER\_PERIOD 100E-3

#define FLAG\_DOWN 0

#define TIMER\_CLOCK\_FREQUENCY (XPAR\_CPU\_CORTEXA9\_0\_CPU\_CLK\_FREQ\_HZ / 2)

#define TIMER\_LOAD\_VALUE ((TIMER\_PERIOD \* TIMER\_CLOCK\_FREQUENCY) - 1.0)

int main()

{

srand(LARGE\_PRIME\_NUMBER);

// Initialize the GPIO LED driver and print out an error message if it fails (argument = true).

// You need to init the LEDs so that LD4 can function as a heartbeat.

leds\_init(true);

// Init all interrupts (but does not enable the interrupts at the devices).

// Prints an error message if an internal failure occurs because the argument = true.

interrupts\_initAll(true);

interrupts\_setPrivateTimerLoadValue(TIMER\_LOAD\_VALUE);

u32 privateTimerTicksPerSecond = interrupts\_getPrivateTimerTicksPerSecond();

printf("private timer ticks per second: %ld\n\r", privateTimerTicksPerSecond);

// Allow the timer to generate interrupts.

interrupts\_enableTimerGlobalInts();

// Initialization of the clock display is not time-dependent, do it outside of the state machine.

//TicTacToeDisplay\_init();

display\_init();

display\_fillScreen(DISPLAY\_BLACK);

buttons\_init();

// Keep track of your personal interrupt count. Want to make sure that you don't miss any interrupts.

int32\_t personalInterruptCount = 0;

// Start the private ARM timer running.

interrupts\_startArmPrivateTimer();

// Enable interrupts at the ARM.

interrupts\_enableArmInts();

// interrupts\_isrInvocationCount() returns the number of times that the timer ISR was invoked.

// This value is maintained by the timer ISR. Compare this number with your own local

// interrupt count to determine if you have missed any interrupts.

while (interrupts\_isrInvocationCount() < (TOTAL\_SECONDS \* privateTimerTicksPerSecond)) {

if (interrupts\_isrFlagGlobal) { // This is a global flag that is set by the timer interrupt handler.

// Count ticks.

double duration0;

personalInterruptCount++;

SimonControl\_tick();

verifySequence\_tick();

flashSequence\_tick();

simonbuttonHandler\_tick();

interrupts\_isrFlagGlobal = FLAG\_DOWN;

}

}

interrupts\_disableArmInts();

printf("isr invocation count: %ld\n\r", interrupts\_isrInvocationCount());

printf("internal interrupt count: %ld\n\r", personalInterruptCount);

return 0;

}

#ifndef BUTTONHANDLER\_H\_

#define BUTTONHANDLER\_H\_

// Get the simon region numbers. See the source code for the region numbering scheme.

uint8\_t simonbuttonHandler\_getRegionNumber();

// Turn on the state machine. Part of the interlock.

void simonbuttonHandler\_enable();

// Turn off the state machine. Part of the interlock.

void simonbuttonHandler\_disable();

// Other state machines can call this function to see if the user has stopped touching the pad.

bool simonbuttonHandler\_releaseDetected();

// Standard tick function.

void simonbuttonHandler\_tick();

// This tests the functionality of the buttonHandler state machine.

// buttonHandler\_runTest(int16\_t touchCount) runs the test until

// the user has touched the screen touchCount times. It indicates

// that a button was pushed by drawing a large square while

// the button is pressed and then erasing the large square and

// redrawing the button when the user releases their touch.

void simonbuttonHandler\_runTest(int16\_t touchCount);

#endif /\* BUTTONHANDLER\_H\_ \*/

//simonButtonHandler.c

#include "simonDisplay.h"

#include<stdio.h>

#include<stdint.h>

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "supportFiles/utils.h"

#include "simonButtonHandler.h"

#define RUN\_TEST\_TERMINATION\_MESSAGE1 "buttonHandler\_runTest()"

#define RUN\_TEST\_TERMINATION\_MESSAGE2 "terminated."

#define RUN\_TEST\_TEXT\_SIZE 2

#define SIMON\_BUTTON\_WIDTH 60

#define SIMON\_DISPLAY\_GETREGION -1

#define SIMON\_DISPLAY\_REGION\_0 0

#define SIMON\_DISPLAY\_REGION\_1 1

#define SIMON\_DISPLAY\_REGION\_2 2

#define SIMON\_DISPLAY\_REGION\_3 3

#define TOUCHES\_INIT

#define PIXEL\_COORD\_ZERO 0

#define ONE\_MS

#define DISPLAY\_WIDTH\_HALVES display\_width()/2

#define DISPLAY\_HEIGHT\_HALVES display\_height()/2

#define AD\_TIMER\_EXPIRED\_VALUE 1

#define AD\_TIMER\_INIT 0

#define RUN\_TEST\_INIT 0

#define BOOL\_INIT false

#define ENABLED true

#define DISABLED false

uint16\_t touches = TOUCHES\_INIT;

uint8\_t region = SIMON\_DISPLAY\_GETREGION;

uint16\_t adTimer = AD\_TIMER\_INIT;

bool enable = false;

bool release = true;

bool erase = false;

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

// simonbuttonHandler(getRegionNumber()

// determines which region of the screen is touched

// converts and returns the touched point into a simon Region Number

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

uint8\_t simonbuttonHandler\_getRegionNumber()

{

int16\_t x;

int16\_t y;

uint8\_t z;

display\_getTouchedPoint(&x,&y,&z);

if (x < PIXEL\_COORD\_ZERO || y < PIXEL\_COORD\_ZERO)

{

return SIMON\_DISPLAY\_GETREGION;

}

if (x < DISPLAY\_WIDTH\_HALVES)

{

if (y < DISPLAY\_HEIGHT\_HALVES)

{

return SIMON\_DISPLAY\_REGION\_0;

}

else

{

return SIMON\_DISPLAY\_REGION\_2;

}

}

else

{

if (y < DISPLAY\_HEIGHT\_HALVES)

{

return SIMON\_DISPLAY\_REGION\_1;

}

else

{

return SIMON\_DISPLAY\_REGION\_3;

}

}

}

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

// simonbuttonHandler\_enable()

// enables the button handler state machine

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

void simonbuttonHandler\_enable()

{

enable = true;

}

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

// simonbuttonHandler\_disable()

// disables the button handler state machine

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

void simonbuttonHandler\_disable()

{

enable = false;

}

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

// simonbuttonHandler\_releaseDetected()

// detects a release of the touch screen

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

bool simonbuttonHandler\_releaseDetected()

{

return release;

}

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

// simonbuttonHandler\_runTest(int16\_t touchCountArt)

// @param: int16\_t touchCountArt

// Runs the button handler until touchCountArg touches are achieved

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

void simonbuttonHandler\_runTest(int16\_t touchCountArg)

{

int16\_t touchCount = RUN\_TEST\_INIT; // Keep track of the number of touches.

display\_init(); // Always have to init the display.

display\_fillScreen(DISPLAY\_BLACK); // Clear the display.

simonDisplay\_drawAllButtons(); // Draw the four buttons.

simonbuttonHandler\_enable();

while (touchCount < touchCountArg) { // Loop here while touchCount is less than the touchCountArg

simonbuttonHandler\_tick(); // Advance the state machine.

utils\_msDelay(ONE\_MS); // Wait here for 1 ms.

if (simonbuttonHandler\_releaseDetected()) { // If a release is detected, then the screen was touched.

touchCount++; // Keep track of the number of touches.

printf("button released: %d\n\r", simonbuttonHandler\_getRegionNumber()); // Get the region number that was touched.

simonbuttonHandler\_disable(); // Interlocked behavior: handshake with the button handler (now disabled).

utils\_msDelay(ONE\_MS); // wait 1 ms.

simonbuttonHandler\_tick(); // Advance the state machine.

simonbuttonHandler\_enable(); // Interlocked behavior: enable the buttonHandler.

utils\_msDelay(ONE\_MS); // wait 1 ms.

simonbuttonHandler\_tick(); // Advance the state machine.

}

}

display\_fillScreen(DISPLAY\_BLACK); // clear the screen.

display\_setTextSize(RUN\_TEST\_TEXT\_SIZE); // Set the text size.

display\_setCursor(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES); // Move the cursor to a rough center point.

display\_println(RUN\_TEST\_TERMINATION\_MESSAGE1); // Print the termination message on two lines.

display\_println(RUN\_TEST\_TERMINATION\_MESSAGE2);

}

enum simonButtonHandlerStates {sbh\_init, sbh\_draw, sbh\_ad\_timer, sbh\_waiting\_for\_release, sbh\_complete} sbh\_state;

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

// simonbuttonHandler\_tick()

// button handler state machine

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

void simonbuttonHandler\_tick()

{

switch(sbh\_state)

{

//sbh\_init

//Initializes values and draws the buttons

case sbh\_init:

release = BOOL\_INIT;

adTimer = AD\_TIMER\_INIT;

region = SIMON\_DISPLAY\_GETREGION;

if(enable == ENABLED)

{

simonDisplay\_drawAllButtons();

sbh\_state = sbh\_ad\_timer;

}

break;

//sbh\_ad\_timer

//Debounces the touches

//If the button handler is not enabled, transitions to completed

case sbh\_ad\_timer:

if(!enable)

{

sbh\_state = sbh\_complete;

}

if(display\_isTouched())

{

adTimer++;

}

if(adTimer == AD\_TIMER\_EXPIRED\_VALUE)

{

sbh\_state = sbh\_draw;

display\_clearOldTouchData();

adTimer = AD\_TIMER\_INIT;

}

break;

//sbh\_draw

//draws the square in the region of the user's touch

case sbh\_draw:

display\_clearOldTouchData();

region = simonbuttonHandler\_getRegionNumber();

simonDisplay\_drawSquare(region, erase);

erase = ENABLED;

release = DISABLED;

sbh\_state = sbh\_waiting\_for\_release;

break;

//sbh\_waiting\_for\_release

//waits for the user to release his/her touch

case sbh\_waiting\_for\_release:

if(!display\_isTouched())

{

simonDisplay\_drawSquare(region, ENABLED);

simonDisplay\_drawButton(region);

erase = DISABLED;

release = ENABLED;

sbh\_state = sbh\_complete;

}

break;

//sbh\_complete

// sets the button handler's completed bool to true

// returns to init if the button handler is enabled

case sbh\_complete:

erase = DISABLED;

if(enable)

{

sbh\_state = sbh\_init;

}

break;

default:

sbh\_state= sbh\_init;

}

}

#ifndef GLOBALS\_H\_

#define GLOBALS\_H\_

#define GLOBALS\_MAX\_FLASH\_SEQUENCE 1000 // Make it big so you can use it for a splash screen.

// This is the length of the complete sequence at maximum length.

// You must copy the contents of the sequence[] array into the global variable that you maintain.

// Do not just grab the pointer as this will fail.

void globals\_setSequence(const uint8\_t sequence[], uint16\_t length);

// This returns the value of the sequence at the index.

uint8\_t globals\_getSequenceValue(uint16\_t index);

// Retrieve the sequence length.

uint16\_t globals\_getSequenceLength();

// This is the length of the sequence that you are currently working on.

void globals\_setSequenceIterationLength(uint16\_t length);

// This is the length of the sequence that you are currently working on (not the maximum length but the interim length as

// the use works through the pattern one color at a time.

uint16\_t globals\_getSequenceIterationLength();

#endif /\* GLOBALS\_H\_ \*/

//globals.c

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "globals.h"

#define STARTING\_SIZE 100

uint16\_t SequenceIterationLength;

uint16\_t SequenceLength;

int8\_t sequencelist[STARTING\_SIZE];

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

//globals\_setSequence(const uint8\_t sequence[], uint16\_t length)

//@param: const uint8\_t sequence[], uint16\_t length

// Sets the global array of sequences

// This is the length of the complete sequence at maximum length.

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

void globals\_setSequence(const uint8\_t sequence[], uint16\_t length)

{

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

{

sequencelist[i] = sequence[i];

}

}

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

// globals\_getSequenceValue(uint16\_t index)

// @param: uint16\_t index

// This returns the value of the sequence at the index.

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

uint8\_t globals\_getSequenceValue(uint16\_t index)

{

return sequencelist[index];

}

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

// globals\_getSequenceLength()

// Retrieve the sequence length.

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

uint16\_t globals\_getSequenceLength()

{

return SequenceLength;

}

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

// globals\_setSequenceIterationLength(uint16\_t length)

// @param: uint16\_t length

// This is the length of the sequence that you are currently working on.

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

void globals\_setSequenceIterationLength(uint16\_t length)

{

SequenceIterationLength = length;

}

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

// globals\_getSequenceIterationLength

// This is the length of the sequence that you are currently working on (not the max length)

// as the player is working through the color sequence

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

uint16\_t globals\_getSequenceIterationLength()

{

return SequenceIterationLength;

}

#ifndef SIMONDISPLAY\_H\_

#define SIMONDISPLAY\_H\_

#include <stdbool.h>

#include <stdint.h>

// Width, height of the simon "buttons"

#define SIMON\_DISPLAY\_BUTTON\_WIDTH 60

#define SIMON\_DISPLAY\_BUTTON\_HEIGHT 60

// Given coordinates from the touch pad, computes the region number.

// The entire touch-screen is divided into 4 rectangular regions, numbered 0 - 3.

// Each region will be drawn with a different color. Colored buttons remind

// the user which square is associated with each color. When you press

// a region, computeRegionNumber returns the region number that is used

// by the other routines.

/\*

|----------|----------|

| | |

| 0 | 1 |

| (RED) | (YELLOW) |

-----------------------

| | |

| 2 | 3 |

| (BLUE) | (GREEN) |

-----------------------

\*/

// These are the definitions for the regions.

#define SIMON\_DISPLAY\_REGION\_0 0

#define SIMON\_DISPLAY\_REGION\_1 1

#define SIMON\_DISPLAY\_REGION\_2 2

#define SIMON\_DISPLAY\_REGION\_3 3

int8\_t simonDisplay\_computeRegionNumber(int16\_t x, int16\_t y);

void simonDisplay\_drawStartScreen(bool erase);

// Draws a colored "button" that the user can touch.

// The colored button is centered in the region but does not fill the region.

void simonDisplay\_drawButton(uint8\_t regionNumber);

// Convenience function that draws all of the buttons.

void simonDisplay\_drawAllButtons();

// Draws a bigger square that completely fills the region.

// If the erase argument is true, it draws the square as black background to "erase" it.

void simonDisplay\_drawSquare(uint8\_t regionNo, bool erase);

// Runs a brief demonstration of how buttons can be pressed and squares lit up to implement the user

// interface of the Simon game. The routine will continue to run until the touchCount has been reached, e.g.,

// the user has touched the pad touchCount times.

// I used a busy-wait delay (utils\_msDelay) that uses a for-loop and just blocks until the time has passed.

// When you implement the game, you CANNOT use this function as we discussed in class. Implement the delay

// using the non-blocking state-machine approach discussed in class.

void simonDisplay\_runTest(uint16\_t touchCount);

void simonDisplay\_eraseAllButtons();

void simonDisplay\_eraseButton(uint8\_t regionNumber);

#endif /\* SIMONDISPLAY\_H\_ \*/

//simonDisplay.c

#include "simonDisplay.h"

#include<stdio.h>

#include<stdint.h>

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "supportFiles/utils.h"

#define SIMON\_BUTTON\_WIDTH 60

#define SIMON\_BUTTON\_WIDTH\_HALVES SIMON\_BUTTON\_WIDTH\_HALVES

#define SIMON\_BUTTON\_HEIGHT 60

#define SIMON\_BUTTON\_HEIGHT\_HALVES SIMON\_BUTTON\_HEIGHT/2

#define ONE\_FOURTH\_HEIGHT display\_height()\*1/4

#define THREE\_FOURTHS\_HEIGHT display\_height()\*3/4

#define ONE\_FOURTH\_WIDTH display\_width()\*1/4

#define THREE\_FOURTHS\_WIDTH display\_width()\*3/4

#define DISPLAY\_WIDTH\_HALVES DISPLAY\_WIDTH\_HALVES

#define DISPLAY\_HEIGHT\_HALVES DISPLAY\_HEIGHT\_HALVES

#define PIXEL\_COORD\_ZERO 0

#define TOUCHES\_INIT 0

#define TOUCH\_PANEL\_ANALOG\_PROCESSING\_DELAY\_IN\_MS 60 // in ms

#define MAX\_STR 255

#define TEXT\_SIZE\_2 2

#define TEXT\_SIZE\_5 5

#define SIMON\_REGION\_0 0

#define SIMON\_REGION\_1 1

#define SIMON\_REGION\_2 2

#define SIMON\_REGION\_3 3

#define INVALID\_REGION -1

#define TOUCH\_PANEL\_ANALOG\_PROCESSING\_DELAY\_IN\_MS 60

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

// simonDisplay\_computeRegionNumber(int16\_t x, int16\_t y)

// @param: int16\_t x, int16\_t y

// Computes the region of the screen that was touched based on the parameters x and y

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

int8\_t simonDisplay\_computeRegionNumber(int16\_t x, int16\_t y)

{

if(x < PIXEL\_COORD\_ZERO || y < PIXEL\_COORD\_ZERO)

{

return INVALID\_REGION;

}

if(x < DISPLAY\_WIDTH\_HALVES)

{

if(y < DISPLAY\_HEIGHT\_HALVES)

{

return SIMON\_REGION\_0;

}

else

{

return SIMON\_REGION\_2;

}

}

else

{

if (y < DISPLAY\_HEIGHT\_HALVES)

{

return SIMON\_REGION\_1;

}

else

{

return SIMON\_REGION\_3;

}

}

}

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

// simonDisplay\_drawButton(uint8\_t regionNumber)

// @param: uint8\_t regionNumber

// Draws a small, colored button in each screen region

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

void simonDisplay\_drawButton(uint8\_t regionNumber)

{

//if the region is invalid, do not draw

if (regionNumber < PIXEL\_COORD\_ZERO)

{

return;

}

switch(regionNumber)

{

case 2:

display\_fillRect(ONE\_WIDTH\_FOURTH-(SIMON\_BUTTON\_WIDTH\_HALVES), (THREE\_FOURTHS\_HEIGHT)-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_BLUE);

break;

case 0:

display\_fillRect((ONE\_FOURTH\_WIDTH)-(SIMON\_BUTTON\_WIDTH\_HALVES), (ONE\_FOURTH\_HEIGHT)-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_RED);

break;

case 1:

display\_fillRect(THREE\_FOURTHS\_WIDTH-(SIMON\_BUTTON\_WIDTH\_HALVES), ONE\_FOURTH\_HEIGHT-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_YELLOW);

break;

case 3:

display\_fillRect(THREE\_FOURTHS\_WIDTH-(SIMON\_BUTTON\_WIDTH\_HALVES), THREE\_FOURTHS\_HEIGHT-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_GREEN);

break;

}

}

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

// simonDisplay\_drawAllButtons()

// draws a button in each of the four regions

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

void simonDisplay\_drawAllButtons()

{

simonDisplay\_drawButton(SIMON\_DISPLAY\_REGION\_0);

simonDisplay\_drawButton(SIMON\_DISPLAY\_REGION\_1);

simonDisplay\_drawButton(SIMON\_DISPLAY\_REGION\_2);

simonDisplay\_drawButton(SIMON\_DISPLAY\_REGION\_3);

}

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

// simonDisplay\_eraseButton(uint8\_t regionNumber)

// @param: uint8\_t regionNumber

// erases all four of the buttons

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

void simonDisplay\_eraseButton(uint8\_t regionNumber)

{

// Do nothing if the region number is negative (illegal region, off LCD screen).

if (regionNumber < PIXEL\_COORD\_ZERO)

{

return;

}

switch(regionNumber)

{

case 2:

display\_fillRect(ONE\_FOURTH\_WIDTH-(SIMON\_BUTTON\_WIDTH\_HALVES), (THREE\_FOURTHS\_HEIGHT)-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_BLACK);

break;

case 0:

display\_fillRect((ONE\_FOURTH\_WIDTH)-(SIMON\_BUTTON\_WIDTH\_HALVES), (ONE\_FOURTH\_HEIGHT)-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_BLACK);

break;

case 1:

display\_fillRect(3\*ONE\_FOURTH\_WIDTH-(SIMON\_BUTTON\_WIDTH\_HALVES), ONE\_FOURTH\_HEIGHT-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_BLACK);

break;

case 3:

display\_fillRect(3\*ONE\_FOURTH\_WIDTH-(SIMON\_BUTTON\_WIDTH\_HALVES), THREE\_FOURTHS\_HEIGHT-(SIMON\_BUTTON\_WIDTH\_HALVES), SIMON\_BUTTON\_WIDTH, SIMON\_BUTTON\_HEIGHT, DISPLAY\_BLACK);

break;

}

}

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

// simonDisplay\_eraseAllButtons()

// erases all of the buttons

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

void simonDisplay\_eraseAllButtons()

{

simonDisplay\_eraseButton(SIMON\_DISPLAY\_REGION\_0);

simonDisplay\_eraseButton(SIMON\_DISPLAY\_REGION\_1);

simonDisplay\_eraseButton(SIMON\_DISPLAY\_REGION\_2);

simonDisplay\_eraseButton(SIMON\_DISPLAY\_REGION\_3);

}

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

// simonDisplay\_drawSquare(uint8\_t regionNo, bool erase)

// @param: uint8\_t regionNo, bool erase

// draws a full square in the corresponding region

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

void simonDisplay\_drawSquare(uint8\_t regionNo, bool erase)

{

// Do nothing if the region number is illegal (off LCD screen).

if (regionNo < PIXEL\_COORD\_ZERO)

{

return;

}

switch(regionNo)

{

case 0:

if (!erase)

{

display\_fillRect(PIXEL\_COORD\_ZERO, PIXEL\_COORD\_ZERO, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_RED);

}

else

{

display\_fillRect(PIXEL\_COORD\_ZERO, PIXEL\_COORD\_ZERO, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_BLACK);

}

break;

case 1:

if (!erase)

{

display\_fillRect(DISPLAY\_WIDTH\_HALVES, PIXEL\_COORD\_ZERO, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_YELLOW);

}

else

{

display\_fillRect(DISPLAY\_WIDTH\_HALVES, PIXEL\_COORD\_ZERO, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_BLACK);

}

break;

case 2:

if (!erase)

{

display\_fillRect(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_BLUE);

}

else

{

display\_fillRect(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_BLACK);

}

break;

case 3:

if (!erase)

{

display\_fillRect(DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_GREEN);

}

else

{

display\_fillRect(DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_WIDTH\_HALVES, DISPLAY\_HEIGHT\_HALVES, DISPLAY\_BLACK);

}

break;

}

}

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

// simonDisplay\_runTest(uint16\_t touchCount)

// @param: uint16\_t touchCount

// Runs a brief demonstration of how buttons can be pressed

//The routine will continue to run until the touchCount has been reached, e.g.,

// the user has touched the pad touchCount times.

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

void simonDisplay\_runTest(uint16\_t touchCount) {

display\_init(); // Always initialize the display.

char str[MAX\_STR]; // Enough for some simple printing.

uint8\_t regionNumber;

uint16\_t touches = TOUCHES\_INIT;

// Write an informational message and wait for the user to touch the LCD.

display\_fillScreen(DISPLAY\_BLACK); // clear the screen.

display\_setCursor(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES); //

display\_setTextSize(TEXT\_SIZE\_2);

display\_setTextColor(DISPLAY\_RED, DISPLAY\_BLACK);

sprintf(str, "Touch and release to start the Simon demo.");

display\_println(str);

display\_println();

sprintf(str, "Demo will terminate after %d touches.", touchCount);

display\_println(str);

while (!display\_isTouched()); // Wait here until the screen is touched.

while (display\_isTouched()); // Now wait until the touch is released.

display\_fillScreen(DISPLAY\_BLACK); // Clear the screen.

simonDisplay\_drawAllButtons(); // Draw all of the buttons.

bool touched = false; // Keep track of when the pad is touched.

int16\_t x, y; // Use these to keep track of coordinates.

uint8\_t z; // This is the relative touch pressure.

while (touches < touchCount) { // Run the loop according to the number of touches passed in.

if (!display\_isTouched() && touched) { // user has stopped touching the pad.

simonDisplay\_drawSquare(regionNumber, true); // Erase the square.

simonDisplay\_drawButton(regionNumber); // DISPLAY\_REDraw the button.

touched = false; // Released the touch, set touched to false.

} else if (display\_isTouched() && !touched) { // User started touching the pad.

touched = true; // Just touched the pad, set touched = true.

touches++; // Keep track of the number of touches.

display\_clearOldTouchData(); // Get rid of data from previous touches.

// Must wait this many milliseconds for the chip to do analog processing.

utils\_msDelay(TOUCH\_PANEL\_ANALOG\_PROCESSING\_DELAY\_IN\_MS);

display\_getTouchedPoint(&x, &y, &z); // After the wait, get the touched point.

regionNumber = simonDisplay\_computeRegionNumber(x, y);// Compute the region number.

simonDisplay\_drawSquare(regionNumber, false); // Draw the square (erase = false).

}

}

// Done with the demo, write an informational message to the user.

display\_fillScreen(DISPLAY\_BLACK); // clear the screen.

display\_setCursor(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES); // Place the cursor in the middle of the screen.

display\_setTextSize(TEXT\_SIZE\_2); // Make it readable.

display\_setTextColor(DISPLAY\_RED, DISPLAY\_BLACK); // red is foreground color, black is background color.

sprintf(str, "Simon demo terminated"); // Format a string using sprintf.

display\_println(str); // Print it to the LCD.

sprintf(str, "after %d touches.", touchCount); // Format the rest of the string.

display\_println(str); // Print it to the LCD.

}

#ifndef simonFlashSequence\_H\_

#define simonFlashSequence\_H\_

// Turns on the state machine. Part of the interlock.

void flashSequence\_enable();

// Turns off the state machine. Part of the interlock.

void flashSequence\_disable();

// Other state machines can call this to determine if this state machine is finished.

bool flashSequence\_completed();

// Standard tick function.

void flashSequence\_tick();

// Tests the flashSequence state machine.

void flashSequence\_runTest();

#endif

//simonFlashSequence.c

#include "simonFlashSequence.h"

#include "simonDisplay.h"

#include<stdio.h>

#include<stdint.h>

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "supportFiles/utils.h"

#include "globals.h"

#include "simonVerifySequence.h"

#define INITFLASHVALUE 10

#define INIT\_ZERO 0

#define TWO\_SECONDS 2000

#define PIXEL\_COORD\_ZERO 0

#define ONE 1

#define DISPLAY\_HEIGHT\_HALVES display\_height()/2

#define TEST\_SEQUENCE\_LENGTH 8 // Just use a short test sequence.

uint8\_t flashSequence\_testSequence[TEST\_SEQUENCE\_LENGTH] = {SIMON\_DISPLAY\_REGION\_0,

SIMON\_DISPLAY\_REGION\_1,

SIMON\_DISPLAY\_REGION\_2,

SIMON\_DISPLAY\_REGION\_3,

SIMON\_DISPLAY\_REGION\_3,

SIMON\_DISPLAY\_REGION\_2,

SIMON\_DISPLAY\_REGION\_1,

SIMON\_DISPLAY\_REGION\_0};

#define INCREMENTING\_SEQUENCE\_MESSAGE1 "Incrementing Sequence"

#define RUN\_TEST\_COMPLETE\_MESSAGE "Runtest() Complete"

#define MESSAGE\_TEXT\_SIZE 2

#define FLASH\_HOLD\_EXP 3

#define F\_ENABLED true

#define F\_DISABLED false

bool flash\_enable = false;

bool flash\_completed = false;

bool flash\_erase = false;

uint8\_t flash\_string = INITFLASHVALUE;

uint8\_t init\_value = INIT\_ZERO;

uint16\_t flash\_sequence\_hold = INIT\_ZERO;

uint8\_t current\_sequence\_length;

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

// flashSequence\_enable()

// Turns on the state machine. Part of the interlock.

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

void flashSequence\_enable()

{

flash\_completed = false;

flash\_enable = true;

}

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

//flashSequence\_disable()

// Turns off the state machine. Part of the interlock.

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

void flashSequence\_disable()

{

flash\_enable = false;

}

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

// flashSequence\_completed()

// Other state machines can call this to determine if this state machine is finished.

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

bool flashSequence\_completed()

{

return flash\_completed;

}

enum FlashSequenceStates{flash\_init, flash\_getregion, flash\_hold, flash\_sequence, flash\_complete, flash\_wait} flash\_state;

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

// flashSequence\_tick()

// flash sequence state machine

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

void flashSequence\_tick()

{

switch(flash\_state)

{

//flash\_init

// sets the current sequence length

// resets global bools

case flash\_init:

current\_sequence\_length = globals\_getSequenceIterationLength();

flash\_completed = F\_DISABLED;

if(flash\_enable)

{

flash\_state = flash\_getregion;

}

break;

//flash\_getregion

//draws a square depending on the position of the sequence

case flash\_getregion:

simonDisplay\_drawSquare(globals\_getSequenceValue(init\_value), flash\_erase);

flash\_erase = F\_ENABLED;

flash\_state = flash\_hold;

break;

//flash\_hold

//holds full square on the screen for FLASH\_HOLD\_EXP time

case flash\_hold:

flash\_sequence\_hold++;

if(flash\_sequence\_hold == FLASH\_HOLD\_EXP)

{

init\_value++;

simonDisplay\_drawSquare(globals\_getSequenceValue(init\_value),flash\_erase);

flash\_erase = F\_DISABLED;

flash\_sequence\_hold = INIT\_ZERO;

flash\_state = flash\_sequence;

}

break;

//flash\_sequence

//determines if the sequence length has been reached

case flash\_sequence:

if(init\_value < current\_sequence\_length)

{

flash\_state = flash\_getregion;

}

if(init\_value >= current\_sequence\_length)

{

flash\_state = flash\_complete;

}

break;

//flash\_complete

//reinitialies values and adjusts the completed bool

case flash\_complete:

init\_value = INIT\_ZERO;

flash\_completed = F\_ENABLED;

if(!flash\_enable)

{

flash\_state = flash\_init;

}

break;

default:

flash\_state = flash\_init;

break;

}

}

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

// flashSequence\_printIncrementingMessage()

// Print the incrementing sequence message.

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

void flashSequence\_printIncrementingMessage()

{

display\_fillScreen(DISPLAY\_BLACK);// Otherwise, tell the user that you are incrementing the sequence.

display\_setCursor(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES); // Roughly centered.

display\_println(INCREMENTING\_SEQUENCE\_MESSAGE1); // Print the message.

utils\_msDelay(TWO\_SECONDS); // Hold on for 2 seconds.

display\_fillScreen(DISPLAY\_BLACK); // Clear the screen.

}

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

// flashSequence\_runTest()

// Runs the flash sequence state machine test using pre-determined sequences and lengths

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

void flashSequence\_runTest()

{

display\_init(); // We are using the display.

display\_fillScreen(DISPLAY\_BLACK); // Clear the display.

globals\_setSequence(flashSequence\_testSequence, TEST\_SEQUENCE\_LENGTH); // Set the sequence.

flashSequence\_enable(); // Enable the flashSequence state machine.

int16\_t sequenceLength = ONE; // Start out with a sequence of length 1.

globals\_setSequenceIterationLength(sequenceLength); // Set the iteration length.

display\_setTextSize(MESSAGE\_TEXT\_SIZE); // Use a standard text size.

while (1)

{ // Run forever unless you break.

flashSequence\_tick(); // tick the state machine.

utils\_msDelay(ONE\_MS); // Provide a 1 ms delay.

if(flashSequence\_completed())// When you are done flashing the sequence.

{

flashSequence\_disable(); // Interlock by first disabling the state machine.

}

flashSequence\_tick(); // tick is necessary to advance the state.

utils\_msDelay(ONE\_MS); // don't really need this here, just for completeness.

flashSequence\_enable(); // Finish the interlock by enabling the state machine.

utils\_msDelay(ONE\_MS); // Wait 1 ms for no good reason.

sequenceLength++; // Increment the length of the sequence.

if (sequenceLength > TEST\_SEQUENCE\_LENGTH) // Stop if you have done the full sequence.

{

break;

}

flashSequence\_printIncrementingMessage(); // Tell the user that you are going to the next step

globals\_setSequenceIterationLength(sequenceLength);// Set the length of the pattern.

}

}

// Let the user know that you are finished.

display\_fillScreen(DISPLAY\_BLACK);

display\_setCursor(PIXEL\_COORD\_ZERO, DISPLAY\_HEIGHT\_HALVES);

display\_println(RUN\_TEST\_COMPLETE\_MESSAGE);

}

#ifndef VERIFYSEQUENCE\_H\_

#define VERIFYSEQUENCE\_H\_

enum verifySequence\_infoMessage\_t

{

user\_time\_out\_e, // means that the user waited too long to tap a color.

user\_wrong\_sequence\_e, // means that the user tapped the wrong color.

user\_correct\_sequence\_e, // means that the user tapped the correct sequence.

user\_quit\_e // means that the user wants to quite.

};

// State machine will run when enabled.

void verifySequence\_enable();

// This is part of the interlock. You disable the state-machine and then enable it again.

void verifySequence\_disable();

// Used to detect if there has been a time-out error.

bool verifySequence\_isTimeOutError();

// Used to detect if the user tapped the incorrect sequence.

bool verifySequence\_isUserInputError();

// Used to detect if the verifySequence state machine has finished verifying.

bool verifySequence\_isComplete();

// Standard tick function.

void verifySequence\_tick();

// Standard runTest function.

void verifySequence\_runTest();

void verifySequence\_printInfoMessage(verifySequence\_infoMessage\_t messageType);

void verifySequence\_eraseInfoMessage(verifySequence\_infoMessage\_t messageType);

#endif /\* VERIFYSEQUENCE\_H\_ \*/

//simonVerifySequence.c

#include "simonVerifySequence.h"

#include <stdio.h>

#include "supportFiles/leds.h"

#include "supportFiles/globalTimer.h"

#include "supportFiles/interrupts.h"

#include <stdbool.h>

#include <stdint.h>

#include "clockControl.h"

#include "clockDisplay.h"

#include "supportFiles/display.h"

#include "xparameters.h"

#include "globals.h"

#include "simonDisplay.h"

#include "simonButtonHandler.h"

#include "buttons.h"

#include "supportFiles/utils.h"

#define MESSAGE\_X 0

#define MESSAGE\_Y (display\_width()/4)

#define MESSAGE\_TEXT\_SIZE 2

#define MESSAGE\_STARTING\_OVER

#define TIMER\_EXPIRED\_VALUE 10

#define MESSAGE\_WAIT\_MS 4000 // Display messages for this long.

#define MAX\_TEST\_SEQUENCE\_LENGTH 4

#define INIT\_ZERO 0

#define INVALID\_REGION -1

#define BTN0 1

#define V\_DISABLED false

#define V\_ENABLED true

#define ONE 1

#define ONE\_MS 1

#define ZERO 0

uint8\_t verifySequence\_testSequence[MAX\_TEST\_SEQUENCE\_LENGTH] = {0, 1, 2, 3};

bool verify\_enable = false;

bool verify\_completed = false;

bool verify\_isTimeOutError =false;

bool verify\_isUserInputError= false;

uint16\_t timer\_waiting = INIT\_ZERO;

uint8\_t simonRegion = INVALID\_REGION;

uint16\_t verifyValue = INIT\_ZERO;

uint16\_t verify\_current\_sequence\_length;

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

// verifySequence\_enable()

// enables the verify sequence state machine

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

void verifySequence\_enable()

{

verify\_enable = true;

}

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

// verifySequence\_disable()

// disables the verify sequence state machine, erases the simon buttons

// disables the button handler state machine

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

void verifySequence\_disable()

{

simonbuttonHandler\_disable();

simonDisplay\_eraseAllButtons();

verify\_enable = false;

}

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

// verifySequence\_isTimeOutError

// Used to detect if there has been a time-out error.

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

bool verifySequence\_isTimeOutError()

{

return verify\_isTimeOutError;

}

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

// verifySequence\_isUserInputError()

// Used to detect if the user tapped the incorrect sequence.

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

bool verifySequence\_isUserInputError()

{

return verify\_isUserInputError;

}

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

// verifySequence\_isComplete()

// Used to detect if the verifySequence state machine has finished verifying.

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

bool verifySequence\_isComplete()

{

return verify\_completed;

}

enum VerifyStates {verify\_init, verify\_waiting, verify\_wait\_release, verify\_validate, verify\_complete, final\_state} verify\_state;

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

// verifySequence\_tick()

// verify sequence state machine

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

void verifySequence\_tick()

{

switch(verify\_state)

{

//verify\_init

//Resets all error cases and sets all timers to zero

//Gets the current sequence length

//enables the button handler

case verify\_init:

verify\_isTimeOutError = V\_DISABLED;

verify\_isUserInputError = V\_DISABLED;

verify\_completed = V\_DISABLED;

timer\_waiting = INIT\_ZERO;

verifyValue = INIT\_ZERO;

verify\_current\_sequence\_length = globals\_getSequenceIterationLength();

if(verify\_enable)

{

simonbuttonHandler\_enable();

verify\_state = verify\_waiting;

}

break;

//verify\_waiting

//provides the user with time to touch the screen

//if the user does not touch before TIMER\_EXPIRED\_VALUE, verify\_isTimeOutError occurs

//disables the button handler after waiting is over

case verify\_waiting:

timer\_waiting++;

if(timer\_waiting == TIMER\_EXPIRED\_VALUE)

{

verify\_isTimeOutError = V\_ENABLED;

simonbuttonHandler\_disable();

verify\_state = verify\_complete;

}

if(display\_isTouched())

{

if(!simonbuttonHandler\_releaseDetected() && timer\_waiting < TIMER\_EXPIRED\_VALUE)

{

verify\_state = verify\_wait\_release;

}

}

break;

//verify\_wait\_release

//waits for the user to release the touch screen and disables the button handler

case verify\_wait\_release:

if(simonbuttonHandler\_releaseDetected())

{

simonRegion = simonbuttonHandler\_getRegionNumber();

timer\_waiting = INIT\_ZERO;

simonbuttonHandler\_disable();

verify\_state = verify\_validate;

}

break;

//verify\_validate

//compares the verifyValue to the sequence value to see if the sequence is over

//if they are not the same, user input error has occurred

case verify\_validate:

if(simonRegion == globals\_getSequenceValue(verifyValue))

{

verifyValue++;

if(verifyValue == verify\_current\_sequence\_length)

{

verifyValue = INIT\_ZERO;

verify\_state = verify\_complete;

}

else

{

timer\_waiting = INIT\_ZERO;

verify\_state = verify\_waiting;

simonbuttonHandler\_enable();

}

}

else

{

verify\_isUserInputError = V\_ENABLED;

verify\_state = verify\_complete;

}

break;

//verify\_complete

//signifies the state machine has completed

//restarts if and only if the state machine is enabled

case verify\_complete:

verify\_completed = V\_ENABLED;

timer\_waiting = INIT\_ZERO;

if(verify\_enable)

{

verify\_state = verify\_init;

}

break;

default:

verify\_state = verify\_init;

break;

}

}

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

// verifySequence\_printInstructions(uint8\_t length, bool startingOver)

// Prints the instructions that the user should follow when

// testing the verifySequence state machine.

// Takes an argument that specifies the length of the sequence so that

// the instructions are tailored for the length of the sequence.

// This assumes a simple incrementing pattern so that it is simple to

// instruct the user.

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

void verifySequence\_printInstructions(uint8\_t length, bool startingOver)

{

display\_fillScreen(DISPLAY\_BLACK); // Clear the screen.

display\_setTextSize(MESSAGE\_TEXT\_SIZE); // Make it readable.

display\_setCursor(MESSAGE\_X, MESSAGE\_Y); // Rough center.

if (startingOver)// Print a message if you start over.

{

display\_fillScreen(DISPLAY\_BLACK); // Clear the screen if starting over.

display\_setTextColor(DISPLAY\_WHITE); // Print whit text.

display\_println("Starting Over. ");

}

display\_println("Tap: ");

display\_println();

switch (length)

{

case 1:

display\_println("red");

break;

case 2:

display\_println("red, yellow ");

break;

case 3:

display\_println("red, yellow, blue ");

break;

case 4:

display\_println("red, yellow, blue, green ");

break;

default:

break;

}

display\_println("in that order.");

display\_println();

display\_println("hold BTN0 to quit.");

}

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

// incrementSequenceLength(int16\_t sequenceLength)

// @param: int16\_t sequenceLength

// This will set the sequence to a simple sequential pattern.

// Increment the sequence length making sure to skip over 0.

// Used to change the sequence length during the test.

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

int16\_t incrementSequenceLength(int16\_t sequenceLength) {

int16\_t value = (sequenceLength + ONE) % (MAX\_TEST\_SEQUENCE\_LENGTH + ONE);

if (value == ZERO)

{

value++;

}

return value;

}

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

// verifySequence\_printInfroMessage(verifySequence\_infoMessage\_t messageType)

// @param: verifySequence\_infoMessage\_t messageType

// Prints out informational messages based upon a message type (see above).

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

void verifySequence\_printInfoMessage(verifySequence\_infoMessage\_t messageType)

{

display\_setTextColor(DISPLAY\_WHITE);

display\_setCursor(MESSAGE\_X, MESSAGE\_Y);

switch(messageType)

{

case user\_time\_out\_e: // Tell the user that they typed too slowly.

display\_println("Error:");

display\_println();

display\_println(" User tapped sequence");

display\_println(" too slowly.");

break;

case user\_wrong\_sequence\_e: // Tell the user that they tapped the wrong color.

display\_println("Error: ");

display\_println();

display\_println(" User tapped the");

display\_println(" wrong sequence.");

break;

case user\_correct\_sequence\_e: // Tell the user that they were correct.

display\_println("User tapped");

display\_println("the correct sequence.");

break;

case user\_quit\_e: // Acknowledge that you are quitting the test.

display\_println("quitting runTest().");

break;

default:

break;

}

}

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

// verifySequence\_eraseInfroMessage(verifySequence\_infoMessage\_t messageType)

// @param: verifySequence\_infoMessage\_t messageType

// Erases informational messages based upon a message type (see above).

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

void verifySequence\_eraseInfoMessage(verifySequence\_infoMessage\_t messageType)

{

display\_setTextColor(DISPLAY\_BLACK);

display\_setCursor(MESSAGE\_X, MESSAGE\_Y);

switch(messageType)

{

case user\_time\_out\_e: // Tell the user that they typed too slowly.

display\_println("Error:");

display\_println();

display\_println(" User tapped sequence");

display\_println(" too slowly.");

break;

case user\_wrong\_sequence\_e: // Tell the user that they tapped the wrong color.

display\_println("Error: ");

display\_println();

display\_println(" User tapped the");

display\_println(" wrong sequence.");

break;

case user\_correct\_sequence\_e: // Tell the user that they were correct.

display\_println("User tapped");

display\_println("the correct sequence.");

break;

case user\_quit\_e: // Acknowledge that you are quitting the test.

display\_println("quitting runTest().");

break;

default:

break;

}

}

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

// verifySequence\_runTest()

// Tests the verifySequence state machine.

// It prints instructions to the touch-screen. The user responds by tapping the

// correct colors to match the sequence.

// Users can test the error conditions by waiting too long to tap a color or

// by tapping an incorrect color.

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

void verifySequence\_runTest()

{

display\_init(); // Always must do this.

buttons\_init(); // Need to use the push-button package so user can quit.

int16\_t sequenceLength = ONE; // Start out with a sequence length of 1.

verifySequence\_printInstructions(sequenceLength, false); // Tell the user what to do.

utils\_msDelay(MESSAGE\_WAIT\_MS); // Give them a few seconds to read the instructions.

simonDisplay\_drawAllButtons() // Now, draw the buttons.

// Set the test sequence and it's length.

globals\_setSequence(verifySequence\_testSequence, MAX\_TEST\_SEQUENCE\_LENGTH);

globals\_setSequenceIterationLength(sequenceLength);

// Enable the verifySequence state machine.

verifySequence\_enable(); // Everything is interlocked, so first enable the machine.

while (!(buttons\_read() & BTN0))// Need to hold button until it quits as you might be stuck in a delay.

{

// verifySequence uses the buttonHandler state machine so you need to "tick" both of them.

verifySequence\_tick(); // Advance the verifySequence state machine.

simonbuttonHandler\_tick(); // Advance the buttonHandler state machine.

utils\_msDelay(ONE\_MS); // Wait 1 ms.

// If the verifySequence state machine has finished, check the result, otherwise just keep ticking both machines.

if (verifySequence\_isComplete())

{

if (verifySequence\_isTimeOutError())

{ // Was the user too slow?

verifySequence\_printInfoMessage(user\_time\_out\_e); // Yes, tell the user that they were too slow.

}

else if (verifySequence\_isUserInputError())

{ // Did the user tap the wrong color?

verifySequence\_printInfoMessage(user\_wrong\_sequence\_e); // Yes, tell them so.

}

else

{

verifySequence\_printInfoMessage(user\_correct\_sequence\_e); // User was correct if you get here.

}

utils\_msDelay(MESSAGE\_WAIT\_MS); // Allow the user to read the message.

sequenceLength = incrementSequenceLength(sequenceLength); // Increment the sequence.

globals\_setSequenceIterationLength(sequenceLength); // Set the length for the verifySequence state machine.

verifySequence\_printInstructions(sequenceLength, V\_ENABLED); // Print the instructions.

utils\_msDelay(MESSAGE\_WAIT\_MS); // Let the user read the instructions.

verifySequence\_drawButtons(); // Draw the buttons.

verifySequence\_disable(); // Interlock: first step of handshake.

verifySequence\_tick(); // Advance the verifySequence machine.

utils\_msDelay(ONE\_MS); // Wait for 1 ms.

verifySequence\_enable(); // Interlock: second step of handshake.

utils\_msDelay(ONE\_MS); // Wait 1 ms.

}

}

verifySequence\_printInfoMessage(user\_quit\_e); // Quitting, print out an informational message.

}

#ifndef SIMONCONTROL\_H\_

#define SIMONCONTROL\_H\_

#include <stdbool.h>

#include <stdint.h>

#include "globals.h"

#include "simonDisplay.h"

#include "simonButtonHandler.h"

#include "simonFlashSequence.h"

#include "simonVerifySequence.h"

#include <stdio.h>

void SimonControl\_tick();

#endif

//SimonControl.c

#include "SimonControl.h"

#include "supportFiles/display.h"

#define INIT\_ZERO 0

#define CONTROL\_FOUR 4

#define CONTROL\_ONE 1

#define TEXT\_SIZE\_TWO 2

#define TEXT\_SIZE\_FOUR 4

#define NEW\_LEVEL\_SIZE 25

#define SCORE\_MESSAGE\_SIZE 20

#define NEW\_LEVEL\_AD\_TIMER\_EXPIRED 10

#define FINAL\_DELAY\_EXPIRED 2

#define DISPLAY\_SCORE\_TIMER\_EXPIRED 40

#define SIMON\_AD\_TIMER\_EXPIRED 1

#define WAIT\_FOR\_PLAYER\_TIMER\_EXPIRED 30

#define SEQUENCE\_SIZE 50

#define LAST\_DELAY\_EXPIRED 10

#define WAITING\_AD\_TIMER 20

#define CURSOR\_WIDTH display\_width()/3.5

#define CURSOR\_HEIGHT display\_height()/3

#define DISPLAY\_WIDTH\_EIGHTS display\_width()/8

#define DISPLAY\_HEIGHT\_HALVES display\_height()/2

#define DISPLAY\_WIDTH\_FIFTHS display\_width()/5

#define DISPLAY\_HEIGHT\_THIRDS display\_width()/3

#define NEXT\_LINE\_WIDTH display\_width()/7

#define NEXT\_LINE\_HEIGHT display\_height()/3 \* 1.5

#define YAY\_TIMER\_EXPIRED 5

#define LC\_EXPIRED 5

#define SIMON "Simon"

#define PROMPT\_TOUCH "touch to start game"

#define CONGRATULATIONS "Congratulations!"

#define YAY "Yay!"

uint8\_t simonControl\_sequence[SEQUENCE\_SIZE];

uint16\_t sequence\_length = SEQUENCE\_SIZE;

int16\_t wait\_for\_player\_timer = 0;

int16\_t simon\_ad\_timer = 0;

int16\_t display\_score\_timer = 0;

int16\_t final\_delay = 0;

int16\_t new\_level\_ad\_timer = 0;

int16\_t level = 1;

int16\_t squares\_per\_level = 4;

uint16\_t length = 1;

uint16\_t last\_delay = 0;

uint16\_t waiting\_ad\_timer = 0;

uint16\_t sc\_yay\_count = 0;

uint16\_t pread = 0;

uint16\_t lc = 0;

uint16\_t max\_length = 0;

bool screen\_erase = true;

bool screen\_display = false;

bool show\_score = false;

char newLevel\_message[NEW\_LEVEL\_SIZE];

char score\_message[SCORE\_MESSAGE\_SIZE];

enum simonControl\_States {sc\_init, sc\_waiting, sc\_flash\_pause, sc\_debounce, sc\_ad\_timer, sc\_yay\_state, sc\_flash, sc\_wait\_touch, sc\_verify, sc\_new\_level, sc\_new\_level\_ad\_timer, sc\_display\_score,sc\_score\_running, sc\_game\_over, sc\_last\_delay, sc\_last\_score} sc\_state;

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

//@start\_message(bool erase)

//@param: bool erase

//Prints a starting message to the touch screen

//If erase is false, the start message will be printed in white

//If erase is true, the start message will be printed over in black

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

void start\_message(bool erase)

{

if(erase == false)

{

display\_setTextColor(DISPLAY\_WHITE);

}

else

{

display\_setTextColor(DISPLAY\_BLACK);

}

display\_setCursor(CURSOR\_WIDTH, CURSOR\_HEIGHT);

display\_setTextColor(DISPLAY\_WHITE);

display\_setTextSize(TEXT\_SIZE\_FOUR);

display\_println(SIMON);

display\_setCursor(NEXT\_LINE\_WIDTH, NEXT\_LINE\_HEIGHT);

display\_setTextSize(TEXT\_SIZE\_TWO);

display\_println(PROMPT\_TOUCH);

}

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

//@new\_level\_message(bool erase)

//@param: bool erase

//Prints a starting message to the touch screen

//If erase is false, the new level message will be printed in white

//If erase is true, the new level message will be printed over in black

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

void new\_level\_message(bool erase)

{

if(erase == false)

{

display\_setTextColor(DISPLAY\_WHITE);

}

else

{

display\_setTextColor(DISPLAY\_BLACK);

}

display\_setCursor(DISPLAY\_WIDTH\_FIFTHS, DISPLAY\_HEIGHT\_THIRDS );

display\_setTextSize(TEXT\_SIZE\_TWO);

display\_println(CONGRATULATIONS);

display\_setCursor(NEXT\_LINE\_WIDTH, NEXT\_LINE\_HEIGHT);

display\_setTextSize(TEXT\_SIZE\_TWO);

sprintf(newLevel\_message, "touch to start level %2u", level);

display\_println(newLevel\_message);

}

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

//@display\_yay(bool erase)

//@param: bool erase

//Prints a starting message to the touch screen

//If erase is false, Yay! will be printed in white

//If erase is true, Yay! will be printed over in black

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

void display\_yay(bool erase)

{

if(erase)

{

display\_setTextColor(DISPLAY\_BLACK);

}

else

{

display\_setTextColor(DISPLAY\_WHITE);

}

display\_setCursor(DISPLAY\_WIDTH\_FIFTHS, DISPLAY\_HEIGHT\_THIRDS);

display\_setTextSize(TEXT\_SIZE\_FOUR);

display\_println(YAY);

}

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

//@show\_score\_message(bool erase)

//@param: bool erase

//Prints a starting message to the touch screen

//If erase is false, the end level score message will be printed in white

//If erase is true, the end level message will be printed over in black

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

void show\_score\_message(bool erase)

{

if(erase)

{

display\_setTextColor(DISPLAY\_BLACK);

}

else

{

display\_setTextColor(DISPLAY\_WHITE);

}

display\_setCursor(DISPLAY\_WIDTH\_EIGHTS, DISPLAY\_HEIGHT\_HALVES);

display\_setTextSize(TEXT\_SIZE\_TWO);

sprintf(score\_message, "Longest sequence: %2u", length);

display\_println(score\_message);

}

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

//@show\_lc(bool erase)

//@param: bool erase

//Prints a starting message to the touch screen

//If erase is false, the longest count message will be printed in white

//If erase is true, the longest count message will be printed over in black

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

void show\_lc(bool erase)

{

if(erase)

{

display\_setTextColor(DISPLAY\_BLACK);

}

else

{

display\_setTextColor(DISPLAY\_WHITE);

}

display\_setCursor(DISPLAY\_WIDTH\_EIGHTS, DISPLAY\_HEIGHT\_HALVES);

display\_setTextSize(TEXT\_SIZE\_TWO);

sprintf(score\_message, "Longest sequence: %2u", max\_length);

display\_println(score\_message);

}

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

//@screateRandomArray()

// Creates a random number using the rand()

// The array is then set to be the global sequence array

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

void createRandomArray()

{

for(int i = INIT\_ZERO; i < SEQUENCE\_SIZE; i++)

{

int16\_t random = rand() % CONTROL\_FOUR;

simonControl\_sequence[i] = random;

}

globals\_setSequence(simonControl\_sequence, sequence\_length);

}

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

//@adjustMaxLength(uint16\_t num)

//@param: uint16\_t num

//Maximizes the max\_length integer variable to be printed if the game ends prematurely

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

void adjustMaxLength(uint16\_t num)

{

if(num > max\_length)

{

max\_length = num;

}

}

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

//@SimonControl\_tick()

//The Control State Machine

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

void SimonControl\_tick()

{

switch(sc\_state)

{

//sc\_init

//Initializes the amount of correct presses to zero

//Disables the state machines so they do not incorrectly run

//Displays a start screen

case sc\_init:

max\_length = INIT\_ZERO;

createRandomArray();

globals\_setSequenceIterationLength(length);

verifySequence\_disable();

flashSequence\_disable();

start\_message();

sc\_state = sc\_waiting;

break;

//sc\_waiting

//Transitions to the ad\_timer debouncer when the screen is touched

case sc\_waiting:

if(display\_isTouched())

{

display\_clearOldTouchData();

sc\_state = sc\_ad\_timer;

}

break;

//sc\_ad\_timer

//Adds a debounce value for screen presses. Once the value is achieved, it transitions the flash state

//Else, it returns to the sc\_waiting state

case sc\_ad\_timer:

simon\_ad\_timer++;

wait\_for\_player\_timer = INIT\_ZERO;

if(display\_isTouched() && simon\_ad\_timer == SIMON\_AD\_TIMER\_EXPIRED)

{

verifySequence\_disable();

start\_message(screen\_erase);

flashSequence\_enable();

sc\_state = sc\_flash;

}

else if(!display\_isTouched() && simon\_ad\_timer == SIMON\_AD\_TIMER\_EXPIRED )

{

sc\_state = sc\_waiting;

}

break;

//sc\_flash

//Disables the flashSequence state machine and enables the verifySequence state machine

case sc\_flash:

simon\_ad\_timer = INIT\_ZERO;

if(flashSequence\_completed())

{

flashSequence\_disable();

verifySequence\_enable();

sc\_state = sc\_verify;

}

break;

//sc\_verify

//Once the verifySequence is complete, a transition will occur based on the conditions of verifyComplete

//If the user does not press fast enough or presses incorrectly, game over occurs

//If the user completes a level, the yay state occurs

//If the user completes a sequence and the level is not over, the next flash sequence occurs

case sc\_verify:

if(verifySequence\_isComplete() && (verifySequence\_isTimeOutError() || verifySequence\_isUserInputError()) )

{

verifySequence\_disable();

sc\_state = sc\_game\_over;

}

else if(verifySequence\_isComplete() && globals\_getSequenceIterationLength() == squares\_per\_level)

{

level++;

verifySequence\_disable();

display\_yay(screen\_display);

squares\_per\_level++;

sc\_state = sc\_yay\_state;

}

else if(verifySequence\_isComplete() && globals\_getSequenceIterationLength() != squares\_per\_level)

{

length++;

verifySequence\_disable();

flashSequence\_enable();

adjustMaxLength(length);

globals\_setSequenceIterationLength(length);

sc\_state = sc\_flash;

}

break;

//sc\_yay\_state

//Congratulates the user on completing a level

//The yay message is displayed up until sc\_yay\_count equals YAY\_TIMER\_EXPIRED

//The state machine transitions to a new level

case sc\_yay\_state:

sc\_yay\_count++;

if(sc\_yay\_count == YAY\_TIMER\_EXPIRED)

{

display\_yay(screen\_erase);

sc\_state = sc\_new\_level;

}

break;

//sc\_new\_level

//Offers the user a chance to play a new, longer level

case sc\_new\_level:

sc\_yay\_count = INIT\_ZERO;

new\_level\_message(display\_screen);

sc\_state = sc\_wait\_touch;

break;

//sc\_wait\_touch

//If the user accepts a new level before the time expires, the game will continue

//else, the final score state is transitioned to

case sc\_wait\_touch:

wait\_for\_player\_timer++;

if(wait\_for\_player\_timer == WAIT\_FOR\_PLAYER\_TIMER\_EXPIRED)

{

new\_level\_message(screen\_erase);

sc\_state = sc\_display\_score;

}

if(display\_isTouched())

{

createRandomArray();

sc\_state = sc\_new\_level\_ad\_timer;

}

break;

//sc\_display\_score

//Reinitializes various timer values for a new game

//Shows the score

case sc\_display\_score:

wait\_for\_player\_timer = INIT\_ZERO;

display\_score\_timer = INIT\_ZERO;

squares\_per\_level = CONTROL\_FOUR;

level = CONTROL\_ONE;

show\_score\_message(screen\_display);

sc\_state = sc\_score\_running;

break;

//sc\_score\_running

//Displays the score for a given amount of time

//Returns to the init state

case sc\_score\_running:

display\_score\_timer++;

if(display\_score\_timer == DISPLAY\_SCORE\_TIMER\_EXPIRED)

{

length = CONTROL\_ONE;

show\_score\_message(screen\_erase);

sc\_state = sc\_init;

}

break;

//sc\_new\_level\_ad\_timer

//Debounces the touch for the user to go to a new level

//Once touched, the game will begin in the flash state

case sc\_new\_level\_ad\_timer:

new\_level\_ad\_timer++;

if(display\_isTouched())

{

new\_level\_message(true);

length = CONTROL\_ONE;

globals\_setSequenceIterationLength(length);

flashSequence\_enable();

new\_level\_ad\_timer=INIT\_ZERO;

sc\_state = sc\_flash;

}

else if(!display\_isTouched() && new\_level\_ad\_timer == NEW\_LEVEL\_AD\_TIMER\_EXPIRED)

{

sc\_state = sc\_wait\_touch;

new\_level\_ad\_timer=INIT\_ZERO;

}

break;

//sc\_game\_over

//Resets values

//Displays the corresponding game over screen for a time FINAL\_DELAY\_EXPIRED

case sc\_game\_over:

final\_delay++;

length = CONTROL\_ONE;

if(final\_delay == FINAL\_DELAY\_EXPIRED || verifySequence\_isUserInputError())

{

final\_delay = INIT\_ZERO;

verifySequence\_printInfoMessage(user\_wrong\_sequence\_e);

sc\_state = sc\_last\_delay;

}

if (verifySequence\_isTimeOutError())

{

final\_delay = INIT\_ZERO;

verifySequence\_printInfoMessage(user\_time\_out\_e);

sc\_state = sc\_last\_delay;

}

break;

//sc\_last\_delay

//Holds the game over display on the screen for a time LAST\_DELAY\_EXPIRED

//Shows the score for the longest count during the game

case sc\_last\_delay:

last\_delay++;

if(last\_delay == LAST\_DELAY\_EXPIRED)

{

last\_delay = INIT\_ZERO;

verifySequence\_eraseInfoMessage(user\_wrong\_sequence\_e);

verifySequence\_eraseInfoMessage(user\_time\_out\_e);

show\_lc(display\_screen);

sc\_state = sc\_last\_score;

}

break;

//sc\_last\_score

//Shows up the longest count during the game for time LC\_EXPIRED

case sc\_last\_score:

lc++;

if(lc == LC\_EXPIRED)

{

lc = INIT\_ZERO;

show\_lc(screen\_erase);

sc\_state = sc\_init;

}

break;

default:

sc\_state = sc\_init;

break;

}

}